Douglas Hergert

Microsoft QuickBASIC

Tecniche di programmazione

tecniche nuove

Douglas Hergert

Microsoft QuickBASIC
Tecniche di programmazione

Douglas Hergert

Microsoft QuickBASIC
Tecniche di programmazione

tecniche nuove

Riassunto dei contenuti

Indice

Introduzione

Microsoft QuickBASIC è una versione strutturata del linguaggio di programmazione BASIC, un ambiente sofisticato di sviluppo dei programmi ed un compilatore veloce ed efficiente, il tutto in un unico pacchetto alla portata di tutte le tasche. Questo libro fornisce un'approfondita guida di riferimento per tutti gli elementi del linguaggio QuickBASIC, versione 4.5. Le duecento singole voci del libro comprendono ogni funzione ed istruzione QuickBASIC ed offrono un'ampia varietà di programmi esempio, che aiuteranno ad imparare, rivedere ed esplorare il linguaggio. Gli argomenti sono divisi in sette parti principali:

Parte I: descrive i tipi di dati, le operazioni, le strutture dati ora supportate in QuickBASIC e le istruzioni usate per dichiarare e lavorare con variabili, vettori e record su disco.
Parte II: tratta gli elementi di programmazione strutturata in QuickBASIC, compresi procedure, cicli, condizioni logiche ed altre strutture di controllo.
Parte III: esamina le tecniche di input ed output per la tastiera, lo schermo, la stampante ed altri dispositivi, oltre una completa introduzione allo stile QuickBASIC per la gestione dei file di dati.
Parte IV: descrive le tecniche grafiche per tutti gli adattatori video disponibili per PC IBM e compatibili.
Parte V: prende in esame la vasta libreria delle funzioni incorporate numeriche o che operano sulle stringhe di caratteri.
Parte VI: studia le importanti tecniche per la gestione degli errori e degli eventi critici.
Parte VII: tratta una varietà di operazioni di sistema, comprese le funzioni data e orario, l'utilizzo delle chiamate di sistema DOS, la gestione della memoria e le routine scritte in altri linguaggi.
La struttura del libro prevede due possibili tipi di lettura. Se si desidera studiare tutti gli strumenti offerti dal QuickBASIC riguardanti uno specifico argomento, bisogna scegliere una delle sette parti principali e svolgerla

come una completa esercitazione di programmazione. Altrimenti, se si vogliono semplicemente informazioni veloci su una particolare caratteristica del QuickBASIC, si può consultare direttamente la voce corrispondente, per trovare informazioni dettagliate sulla sintassi, l'utilizzo e gli elementi specifici dell'istruzione o della funzione in questione. Alla fine di questo libro, vi è una completa tabella riassuntiva, organizzata alfabeticamente per aiutare a localizzare immediatamente ogni parola chiave del linguaggio. Oltre alla trattazione di specifici argomenti, questo testo contiene la descrizione di molti progetti di programmazione di dimensioni considerevoli che eseguono numerosi compiti pratici ed utili. Questi progetti vengono presentati sia come esempi di caratteristiche del QuickBASIC sia come programmi che possono essere studiati, rivisti ed effettivamente utilizzati. Pezzi di questi programmi compaiono nel corso del libro, ma per comodità i listati completi dei programmi sono stati raggruppati nella Parte VIII. Ogni listato è presentato insieme a brevi descrizioni della struttura e della modalità d'esecuzione del programma stesso. Ecco una lista dei programmi presentati nella Parte VIII:

- Il programma *Compleanno* lavora con le informazioni cronologiche relative al personale di una ditta e fornisce esempi dei tipi di dati e strutture di dati del QuickBASIC, compresi vettori, record e vettori di record.
- Il programma *Mese* visualizza su schermo il calendario di uno o più mesi e lo invia alla stampante. Questo programma ha il compito di illustrare gli elementi della programmazione strutturata in QuickBASIC, comprese procedure, cicli e condizioni logiche.
- Il programma *Agenda* crea e gestisce un calendario personale di appuntamenti, memorizzato su disco come database ad accesso diretto. Nel processo, il programma indica l'utilizzo delle istruzioni per gestire file dati ad accesso sequenziale e diretto.
- Il programma *GestioneDate* è un interfaccia utente che permette l'accesso istantaneo alle funzionalità del programma *Agenda*. *GestioneDate* contiene esempi interessanti di tecniche per gestire gli eventi.
- Il programma *DateStoriche* è un gioco a quiz, che mette alla prova le conoscenze storiche del giocatore. Illustra gli strumenti grafici della

potente libreria di funzioni del QuickBASIC e le tecniche per produrre molti effetti sonori.

Oltre a questi principali esempi di programma, nel corso del libro si trovano dozzine di programmi minori, atti ad esporre specifiche istruzioni e funzioni.

Parte I

Tipi di dati, operazioni, variabili e strutture dati

La Parte I è un'introduzione a tre fondamentali argomenti di programmazione:

1. I *tipi di dati* supportati dal linguaggio QuickBASIC.
2. Le *operazioni* che un programma può eseguire su valori appartenenti a particolari tipi di dati.
3. L'utilizzo di *variabili* per contenere valori singoli o multipli in un programma QuickBASIC.

QuickBASIC definisce una varietà significativa di tipi di dati per rappresentare le informazioni ed un buon numero di operazioni che agiscono su tipi specifici di dati. Il Capitolo 1 introduce questi due argomenti essenziali. Gli altri tre capitoli della Parte I descrivono le istruzioni QuickBASIC che creano le variabili e le strutture di dati e lavorano con esse. Le variabili sono nomi che indicano i dati di un programma. La visibilità di una variabile indica la porzione del programma in cui la variabile è utilizzabile. Le voci elencate nel Capitolo 2 illustrano le istruzioni del QuickBASIC che possono essere utilizzate per dichiarare il tipo ed il campo di validità di una variabile. Inoltre, la voce CONST nel Capitolo 2 descrive le *costanti simboliche*, i nomi che rappresentano valori che non cambiano nel corso del programma. Il Capitolo 3 introduce l'uso di *strutture di dati* in QuickBASIC. Una va-

riabile semplice contiene un valore alla volta; al contrario, una struttura contiene contemporaneamente più valori. QuickBASIC supporta una grande varietà di strutture dati.

- Un *vettore* è una variabile che rappresenta una lista, una tabella o un'altra struttura di dati, tutti appartenenti allo stesso tipo. L'istruzione DIM definisce le dimensioni di un vettore.
- Una variabile *record* contiene più elementi detti *campi* che possono appartenere a diversi tipi di dati. Per creare una variabile di questo tipo sono necessarie due operazioni: inizialmente un'istruzione TYPE delinea la struttura del record *in base alle esigenze dell'utente*, poi una dichiarazione tipo DIM crea una variabile record appartenente al tipo definito.
- Un *vettore di record* è una lista o una tabella di record. Ancora una volta, un'istruzione TYPE definisce la struttura del record e poi un'istruzione DIM crea un vettore in cui ogni elemento è un record appartenente al tipo precedentemente definito. Un vettore di record è la struttura ideale per rappresentare in un programma i record di un database.

Il Capitolo 3 spiega come queste tre strutture dati vanno definite ed utilizzate. Variabili, vettori, record e costanti simboliche hanno tutti nomi inventati da chi scrive il programma. Le regole per scrivere i nomi in QuickBASIC sono semplici: un nome inizia con una lettera dell'alfabeto ed è formato al massimo da 40 lettere e numeri (è ammesso anche il punto fermo). Inoltre, il nome di una variabile può terminare con un carattere speciale che indica il tipo di valori che la variabile può contenere: $ per una variabile stringa; % per un numero intero; & per un intero lungo; ! (o nessun carattere speciale) per una variabile a virgola mobile in precisione semplice; e # per una variabile a virgola mobile in doppia precisione. QuickBASIC offre anche altre tecniche per definire il tipo di una variabile, come spiegato nei Capitoli 2 e 3. Infine un assegnamento è un'istruzione che memorizza un valore in una variabile. Il Capitolo 4 descrive una varietà di modi in cui gli assegnamenti possono essere eseguiti.

Il primo programma d'esempio presentato nella Parte VIII, Capitolo 28, chiamato *Compleanno*, illustra tutti questi argomenti nel contesto di una semplice ma pratica applicazione commerciale. Si possono trovare vari estratti del programma *Compleanno* discussi nel corso della Parte I.

Capitolo 1

Tipi di dati ed operazioni

Questo capitolo descrive singolarmente ciascuno dei tipi di dati, disponibili in QuickBASIC, per rappresentare valori sia numerici sia alfabetici. Vengono esaminate le loro caratteristiche, nonché i simboli speciali e le parole chiave usate per definire le variabili dei diversi tipi. (Gli ultimi capitoli della Parte I contengono molte altre informazioni sulla dichiarazione delle variabili e sui tipi di dati definiti dall'utente.) Questo capitolo prende in esame anche le operazioni aritmetiche, relazionali, logiche e su stringhe che sono disponibili per manipolare i dati nei programmi QuickBASIC. Molti termini importanti ricorrono spesso nelle voci di questo capitolo: ad esempio, un'operazione è un'azione che viene eseguita su uno o più valori di dati. Un'operazione unaria lavora su un singolo valore; un'operazione binaria lavora su una coppia di valori. Un operatore è il simbolo QuickBASIC che rappresenta un'operazione. Un operando è uno dei valori dei dati interessati da un'operazione. Infine, un'espressione è una singola operazione o una combinazione di operazioni scritta correttamente; il risultato di un'espressione è un valore che appartiene ad un tipo di dati specificato. Un'espressione appare sempre nel contesto di un'istruzione che ne usa il risultato. Le voci elencate in questo capitolo comprendono molti esempi di operazioni ed espressioni. La maggior parte di questi esempi sono frammenti tratti dai programmi esempio presentati nella Parte VIII del libro. Mentre si studiano gli argomenti di programmazione trattati in questo capitolo, si può far riferimento a quei programmi.

TIPI DI DATI

I tipi di dati elementari del QuickBASIC comprendono le stringhe di caratteri, i numeri interi ed i numeri decimali o a virgola mobile. Questo paragrafo descrive il formato dei tipi di dati disponibili per queste tre categorie. Un valore di qualsiasi tipo può essere espresso in un programma come una costante (chiamata anche valore letterale) o tramite il nome della variabile che lo contiene. QuickBASIC supporta molti formati per esprimere valori numerici costanti. Inoltre, esistono molte tecniche per dichiarare delle variabili per ognuno dei tipi di dati elementari. In questa parte del libro, le singole voci danno informazioni circa i formati delle costanti e le modalità delle dichiarazioni di variabili per ogni tipo di dati. (Per ulteriori dettagli sulle dichiarazioni di variabili, vedere il Capitolo 2.) Le caratteristiche essenziali dei tipi di dati numerici sono l'intervallo e la precisione. I tipi interi di QuickBASIC comprendono un intervallo definito dall'insieme dei numeri interi, sempre con una precisione perfetta. I tipi numerici a virgola mobile comprendono intervalli molto più grandi, ma rappresentano i valori con una precisione limitata. Si possono apprendere i dettagli specifici di queste caratteristiche consultando le voci di questa sezione.

Stringhe

Una stringa è una sequenza di caratteri ASCII. Vi sono due tipi di variabili stringa in QuickBASIC: quelle a lunghezza variabile e quelle a lunghezza fissa. Ecco le caratteristiche generali di questi tipi:

Lunghezza massima	32.767 caratteri
Formato dei valori literal	tra virgolette
Suffisso del tipo per i nomi delle variabili	$
Clausola AS (lunghezza variabile)	AS STRING
Clausola AS (lunghezza fissa)	AS STRING * *lungh*

Tradizionalmente, i nomi delle variabili stringa terminano con il simbolo dollaro ($), sebbene QuickBASIC offra altre tecniche per specificare il tipo

delle variabili. Se lo si desidera si può usare un'istruzione DIM, COMMON, SHARED o STATIC con una clausola AS, per dichiarare una stringa a lunghezza variabile oppure una a lunghezza fissa. In particolare, questo è il formato di dichiarazione DIM per una stringa a lunghezza fissa:

```
DIM var AS STRING * lungh
```

L'utilizzo più importante delle stringhe a lunghezza fissa è definire la struttura di una variabile record. Infatti, i campi di tipo stringa in un record devono essere a lunghezza fissa. (Per ulteriori dettagli vedere la voce TYPE.) Un altro modo per definire le variabili stringa è includere un comando DEFSTR all'inizio del programma. DEFSTR indica quali saranno le lettere d'inizio per i nomi delle variabili stringa a lunghezza variabile uscite in quel programma. La funzione incorporata LEN fornisce la lunghezza corrente di una stringa. Per una variabile stringa a lunghezza fissa, LEN dà sempre la lunghezza dichiarata della variabile. Se a una variabile a lunghezza fissa si assegna un valore di lunghezza inferiore a quella, QuickBASIC completa il resto della variabile con degli spazi (codice ASCII 32). Viceversa, se viene assegnato un valore di lunghezza maggiore di quella definita, QuickBASIC tronca il valore assegnato alla lunghezza disponibile. La lunghezza di una variabile stringa a lunghezza variabile, invece, è per definizione il numero di caratteri del valore corrente della variabile stessa. Una stringa nulla ha una lunghezza pari a zero. QuickBASIC supporta due categorie di operatori su stringa. La concatenazione consiste nella giustapposizione di due stringhe. Sulle stringhe si usano gli operatori logici per verificare uguaglianza, disuguaglianza o l'ordine ASCII tra due operandi. Si consulti la parte intitolata "Operazioni con stringhe" per una discussione più approfondita ed esempi. Inoltre, QuickBASIC comprende una vasta gamma di funzioni incorporate che lavorano sulle stringhe. Questo argomento verrà trattato nella Parte V di questo libro.

Tipi interi

Un dato intero è un numero intero positivo o negativo; in questa categoria QuickBASIC supporta due diversi tipi di dati: interi ed interi lunghi. L'intero standard ha un intervallo relativamente limitato di valori possibili:

Intero

Intervallo	-32.768 a 32.767
Formati per i valori literal	Decimale, esadecimale, ottale
Suffisso che indica il tipo per i nomi delle variabili	%
Clausola AS per le dichiarazioni	AS INTEGER
Memoria richiesta per un valore	Due byte

Al contrario, l'intero lungo ha un intervallo dell'ordine dei miliardi, ma richiede uno spazio di memoria doppio per ogni valore:

Intero lungo

Intervallo	-2.147.483.648 a 2.147.483.647
Formati per i valori literal	Decimale, esadecimale, ottale
Suffisso che indica il tipo per i nomi delle variabili	&
Clausola AS per le dichiarazioni	AS LONG
Memoria richiesta per un valore	Quattro byte

I suffissi tradizionali per le dichiarazioni di tipo delle variabili intere sono % per gli interi standard e & per gli interi lunghi, sebbene QuickBASIC offra altre tecniche per dichiarare i tipi di variabile. Ad esempio, per dichiarare entrambi i tipi di variabili intere, si possono usare le istruzioni DIM, COMMON, SHARED o STATIC, con una clausola AS. Inoltre, le istruzioni DEFINT o DEFLNG indicano le lettere d'inizio scelte per le variabili di tipo intero. In QuickBASIC le costanti intere possono presentarsi in formato decimale, ottale o esadecimale. Il formato decimale è quello di default. I formati ottali ed esadecimali richiedono speciali prefissi: &O (o semplicemente &) per una costante ottale e &H per una costante esadecimale. Ad esempio, le tre costanti seguenti rappresentano tutte valori interi uguali:

12345	Formato decimale
&O30071	Formato ottale
&H3039	Formato esadecimale

Gli interi lunghi possono essere espressi in tutti e tre i formati. Queste tre costanti rappresentano tutte il valore intero lungo positivo più grande disponibile:

2147483647

&O17777777777

&H7FFFFFFF

Le funzioni incorporate OCT$ e HEX$ forniscono gli equivalenti ottali ed esadecimali di argomenti interi. (Vedere il Capitolo 20 per ulteriori dettagli.)

Interi come valori logici

In QuickBASIC gli interi possono rappresentare anche valori logici. Il linguaggio non definisce un tipo di dati logico separato. Ciononostante, tutte le operazioni relazionali e logiche danno come risultati i valori logici vero o falso. Internamente QuickBASIC rappresenta il valore falso come 0 e quello vero come -1. Comunque, si può usare qualsiasi intero per rappresentare un valore logico: 0 è falso ed ogni altro intero viene letto come vero. Per ulteriori informazioni, consultare le parti relative alle operazioni relazionali e logiche nelle pagine seguenti di questo capitolo.

Tipi a virgola mobile

Un numero razionale, spesso detto a virgola mobile, può avere cifre significative prima e dopo il punto decimale. I tipi a virgola mobile hanno intervalli di rappresentazione più grandi dei tipi interi; ma l'esattezza del valore a virgola mobile è garantita solo fino ad un numero specificato di cifre. Vi sono due tipi in questa categoria di dati: il tipo a precisione semplice e quello a precisione doppia. Un valore a precisione semplice ha una precisione fino a sette cifre:

Precisione semplice

Valore più piccolo rappresentabile (approssimativamente)	1,4E-45 (+ o -)

Valore più grande rappresentabile (approssimativamente)	3,4E+38 (+ o -)
Precisione	Sette cifre
Formati per i valori literal	Virgola fissa, virgola mobile
Suffisso che indica il tipo per i nomi delle variabili	! (o nessuno)
Clausola AS per le dichiarazioni	AS SINGLE
Memoria richiesta per un valore	Quattro byte

Al contrario, un valore a doppia precisione mette a disposizione fino a 15 cifre significative, ma richiede il doppio della memoria utilizzata per un valore a precisione semplice:

Doppia precisione

Valore più piccolo rappresentabile (approssimativamente)	4,9D-324 (+ o -)
Valore più grande rappresentabile (approssimativamente)	1,79D+308 (+ o -)
Precisione	Quindici cifre
Formati per i valori literal	Virgola fissa, virgola mobile
Suffisso che indica il tipo per i nomi delle variabili	#
Clausola AS per le dichiarazioni	AS DOUBLE
Memoria richiesta per un valore	Otto byte

L'intervallo di rappresentazione di questi due tipi di dati consente di lavorare con numeri positivi e negativi molto grandi e molto piccoli. Il nome di una variabile a precisione semplice può finire con il carattere facoltativo !. (Precisione semplice è il tipo di dati di default attribuito ad una variabile che non ha nessun suffisso per la dichiarazione del tipo.) Il suffisso del tipo per le variabili a doppia precisione è #. In alternativa, si può usare un'istruzione DIM, COMMON, SHARED o STATIC, con una clausola AS, per definire una variabile a virgola mobile. Inoltre, sono disponibili le istruzioni

DEFSNG e DEFDBL per dichiarare le lettere iniziali scelte per le variabili a virgola mobile. Le costanti a virgola mobile possono presentarsi in due diversi formati. Il formato a virgola fissa consiste in una sequenza di cifre con un punto decimale. (Una costante a virgola fissa può terminare anche con un carattere di suffisso che indica il suo tipo:! per la precisione semplice o # per la precisione doppia.) Il formato a virgola mobile è formato da due parti: la mantissa, che rappresenta le cifre significative e l'esponente, la potenza di 10 che indica la posizione del punto decimale nel numero. Nei numeri a precisione semplice, l'esponente è preceduto dalla lettera E. Ecco due esempi, che rappresentano rispettivamente un numero frazionario molto piccolo ed un numero grande:

9,12E-19

1,56E+22

Nelle costanti a doppia precisione la lettera D indica l'esponente; ad esempio:

9,128D-195

1,569D+222

OPERAZIONI SU STRINGHE

Nel QuickBASIC sono disponibili due tipi di operazioni su stringhe: la concatenazione ed il confronto. La concatenazione esegue la giustapposizione degli operandi (due stringhe) e come risultato dà il valore di un'altra stringa. Fornisce anche il confronto sugli operandi stringhe, ma il risultato è un valore logico di tipo vero o falso. I simboli che rappresentano queste operazioni su stringhe sono gli stessi degli operatori che lavorano con i valori numerici:

- Il segno di addizione (+) rappresenta sia la concatenazione tra stringhe sia l'addizione numerica.
- Gli operatori relazionali (=, <>, <, >, <= e >=) eseguono confronti sia tra coppie di stringhe, sia tra coppie di operandi numerici.

Questo paragrafo descrive l'utilizzo di questi operatori relativamente alle stringhe. Nei paragrafi seguenti di questo capitolo verranno trattate le operazioni numeriche corrispondenti.

Concatenazione

La concatenazione consente di aggiungere una stringa alla fine di un'altra. Il risultato è una terza stringa, la cui lunghezza è uguale alla somma delle lunghezze delle stringhe d'origine. L'operatore per la concatenazione è il segno di addizione:

```
str1 + str2
```

Gli operandi di una concatenazione possono essere valori literal tra virgolette, variabili di tipo stringa o chiamate di funzione che ritornano valori di tipo stringa.

Alcuni esempi

La seguente funzione, chiamata *DataCompl$*, è un estratto del programma *Agenda* (presentato nel Capitolo 30). Dati tre argomenti interi che rappresentano una data, la funzione ritorna una stringa di data del tipo "Merc., Lugl. 12, 1990":

```
FUNCTION DataCompl$ (mese%, giorno%, anno%)

   giorno% = NumGior%(mese%, giorno%, anno%)
   strData$ = NumGior$(gio%) + ".," + NumMese$(mese%)

   IF mese% <> 5 THEN strData$ = strData$ + "."
   strData$ = strData$ + STR$(giorno%) + "," + STR$(anno%)
   DataCompl$ = strData$

END FUNCTION
```

Questa funzione concatena stringhe provenienti da diverse fonti per creare la stringa finale della data. Ad esempio, la funzione accede alle abbreviazioni dei nomi del giorno e del mese contenute nel vettore di stringhe *NumGior$*

e *NumMese$*, che sono entrambi definiti altrove nel programma *Agenda*. Le costanti literal forniscono la punteggiatura per separare le parti che costituiscono la stringa della data. Infine, la routine usa la funzione incorporata STR$ per creare le stringhe equivalenti agli interi che identificano il giorno e l'anno:

```
strData$ = strData$ + STR$(giorno%) + "," + STR$(anno%)
```

Si noti che la stringa viene formata in più fasi e memorizzata progressivamente nella variabile *strData$*. La concatenazione finale aggiunge tre stringhe in fondo al valore attuale di *strData$*.

Confronto tra stringhe

Le operazioni relazionali (rappresentate dai simboli =, <>, <, >, <= e >=) possono essere usate per confrontare coppie di stringhe. Un confronto tra stringhe dà come risultato un valore logico vero o falso.

```
str 1 = str 2     ' Test di uguaglianza tra stringhe.
str 1 <> str 2    ' Test di difformità tra stringhe.
str 1 < str 2     ' Confronto tra codici ASCII.
str 1 > str 2     ' Confronto tra codici ASCII.
str 1 <= str 2    ' Test di uguaglianza o precedenza tra codici ASCII.
str 1 >= str 2    ' Test di uguaglianza o precedenza tra codici ASCII.
```

Nel confronto tra stringhe gli operandi possono comparire come valori tra virgolette, variabili di tipo stringa, chiamate di funzione che ritornano stringhe oppure concatenazioni tra stringhe. QuickBASIC esegue queste operazioni confrontando i caratteri posti nelle posizioni corrispondenti dei due operandi. Ogni operazione dà come risultato un valore logico vero o falso:

- Uguaglianza significa che i due operandi hanno lunghezza uguale e contengono identici caratteri in tutte le corrispondenti posizioni.
- "Minore di" significa che il valore del codice ASCII di un particolare carattere è più piccolo del valore del codice ASCII del carattere corrispondente nell'altra stringa.

- “Maggiore di” significa che il valore del codice ASCII di un carattere è più grande di quello del carattere corrispondente nell’altra stringa.

I codici dei caratteri alfabetici sono importanti nel confronto tra stringhe. Le lettere maiuscole da “A” a “Z”, sono rappresentate dai codice ASCII dal 65 al 90, le minuscole, da “a” a “z”, hanno i codici dal 97 al 122. Di conseguenza, una lettera maiuscola è sempre “minore” di una minuscola. Per confrontare due stringhe senza tener conto del fatto che le lettere siano maiuscole o minuscole, bisogna usare le funzioni UCASE$ o LCASE$ del QuickBASIC, per convertire entrambi gli operandi del confronto in maiuscolo o minuscolo. (Per ulteriori dettagli consultare la Parte V.)

Un esempio

I confronti tra stringhe sono particolarmente importanti per le routine di ordinamento nel caso in cui una stringa sia il campo usato come chiave per l’ordinamento stesso. Ad esempio, il seguente passaggio è uno stralcio del programma *Compleanno* (presentato nel Capitolo 28):

```
FOR i% = 1 TO numDip% - 1
   FOR j% = i% + 1 TO numDip%
   scamb% = FALSE

   SELECT CASE tastoClass%

      CASE NAMESORT
         Nomin1$ = staff(i%).Cogn + staff(i%).Nome
         Nomin2$ = staff(j%).Cogn + staff(j%).Nome
         scamb% = Nomin1$ > Nomin2$

      ' Blocchi aggiuntivi CASE, che riguardano
      ' altri tasti di classificazione.

   END SELECT

   IF scamb% THEN SWAP staff(i%), staff(j%)

   NEXT j%
NEXT i%
```

Questa routine ordina un vettore di record chiamato *staff*, usando come campo chiave su cui eseguire l’ordinamento quello che l’utente seleziona

dalla tastiera. In particolare, se l'utente decide di ordinare il database con i nomi in ordine alfabetico, la procedura utilizza una concatenazione dei campi di *Cogn* e *Nome* come campo chiave. Per confrontare ciascun record, il programma calcola i valori di *Nomin1$* e *Nomin2$* (stringhe composte da cognome e nome concatenati) e poi usa la seguente istruzione per determinare se sono in ordine alfabetico o meno:

```
scamb% = Nomin1$ > Nomin2$
```

Questa istruzione assegna il risultato logico del confronto tra stringhe alla variabile intera *scamb%*. Se *scamb%* è vera, il programma scambia le posizioni dei due record:

```
IF scamb% THEN SWAP staff(i%), staff(j%)
```

(Ulteriori informazioni riguardo questo esempio verranno date nella trattazione sul programma *Compleanno* nel Capitolo 28.)

OPERAZIONI ARITMETICHE

L'aritmetica in QuickBASIC comprende le quattro operazioni più comuni (moltiplicazione, divisione, addizione e sottrazione) oltre ad altre tre operazioni che possono essere meno comuni: l'elevamento a potenza, la divisione intera ed il modulo. Le voci di questo paragrafo descrivono tali operazioni e danno brevi esempi di ognuna. Il problema della precedenza degli operatori si ha nelle espressioni che contengono più di un'operazione. QuickBASIC segue un ordine prestabilito di precedenza per valutare più operazioni: l'elevamento a potenza viene eseguito per primo e poi la negazione. QuickBASIC esegue poi le operazioni in quest'ordine: moltiplicazione e divisione, divisione intera, l'operazione MOD ed infine addizione e sottrazione. Si può comunque non tener conto di questo ordine di default mettendo un'operazione data tra parentesi; le espressioni tra parentesi vengono eseguite per prime. Le parentesi possono anche essere inserite dentro altre parentesi. In questo caso, QuickBASIC esegue per prima l'operazione più interna e poi man mano quelle tra le altre parentesi. Inoltre, talvolta si può decidere di inserire le parentesi in un'espressione solo per maggior chiarezza, anche se le parentesi non cambiano l'ordine di default

delle operazioni. In QuickBASIC un altro importante fattore è costituito dall'intervallo consentito per i tipi di dati numerici. Un errore di esecuzione chiamato *traboccamento* si verifica quando il risultato di un'operazione aritmetica supera l'intervallo consentito per gli operandi. Un modo di evitare questo errore è assicurarsi che almeno uno degli operandi appartenga ad un tipo che ha un intervallo di rappresentazione adeguato per il risultato atteso. Dati degli operandi che appartengono a diversi tipi di dati, nel valutare l'espressione QuickBASIC li converte al tipo che consente di calcolare il risultato più preciso. Questi argomenti verranno trattati nelle pagine seguenti.

Elevamento a potenza

L'operatore ^ eleva un numero ad un esponente, nel seguente modo:

base ^ esponente

Base ed esponente possono essere numeri interi o in virgola mobile. Un'espressione con un esponente frazionario calcola la radice di un numero. Ad esempio, la seguente espressione dà la radice cubica del valore memorizzato in x:

```
x ^ (1 / 3)
```

Si notino le parentesi attorno all'esponente; in questo caso sono richieste poiché l'elevamento a potenza ha precedenza sulla divisione. Un tentativo di usare l'elevamento a potenza per trovare la radice di un numero negativo risulta nell'errore di esecuzione detto Illegal Function Call (Chiamata di funzione non consentita). Un esponente negativo dà l'inverso del risultato dell'elevamento al corrispondente esponente positivo. Ad esempio, le due espressioni seguenti danno lo stesso risultato:

```
x ^ -3
1 / x ^ 3
```

Si osservi che nella prima di queste due espressioni non è richiesta nessuna parentesi. QuickBASIC riconosce la sequenza di due operatori (^ -) come un esponente negativo.

Esempio

Il seguente breve programma dimostra l'operazione di elevamento a potenza:

```
DEFINT I
DEFLNG N
DEFDBL R

CLS
PRINT "Dimostrazione dell'Operatore ^"
PRINT
num = 2 ^ 24
FOR i = 2 TO 24
   rad = num ^ (1 / i)
   IF INT(rad) = rad THEN
      PRINT num; "^ ";
      PRINT USING "(1/##) = "; i;
      PRINT USING "#####"; rad
   END IF
NEXT i
```

Il programma calcola una potenza di 2 e memorizza il risultato nella variabile lunga intera *num*. Poi un ciclo FOR calcola le radici di *num* e visualizza sullo schermo le radici intere. Ecco il risultato che si ottiene:

```
Dimostrazione dell'Operatore ^

   16777216 ^ (1/ 2) = 4096
   16777216 ^ (1/ 3) = 256
   16777216 ^ (1/ 4) = 64
   16777216 ^ (1/ 6) = 16
   16777216 ^ (1/ 8) = 8
   16777216 ^ (1/12) = 4
   16777216 ^ (1/24) = 2
```

Negazione

La negazione è un'operazione unaria che compare in questo formato:

```
-num
```

L'operando, *num*, può essere un valore numerico literal o una variabile. La negazione equivale a moltiplicare un numero per un valore pari a -1. Ha la

precedenza su tutte le altre operazioni aritmetiche, ad eccezione dell'elevamento a potenza. (Comunque, QuickBASIC riconosce la sequenza ^- come un esponente negativo. Vedere la voce "Elevamento a potenza" per ulteriori dettagli.)

Un esempio

La seguente funzione, chiamata *Lim%*, serve per arrotondare un argomento numerico al successivo valore assoluto più alto:

```
FUNCTION Lim% (num)
   IF num <= 0 THEN
      Lim% = FIX(num)
   ELSE
      Lim% = ABS(INT(-num))
   END IF
END FUNCTION
```

L'assegnamento nel blocco ELSE illustra l'operazione di negazione.

```
Lim% = ABS(INT(-num))
```

In questa istruzione, l'argomento inviato alla funzione INT è la negazione del valore memorizzato in *num*. (Vedere la voce FIX per approfondimenti sull'esempio *Lim%*.)

Moltiplicazione

L'operatore * rappresenta la moltiplicazione in QuickBASIC e si usa per calcolare il prodotto di due numeri:

```
val1 * val2
```

Gli operandi possono essere numeri in virgola mobile o interi. Bisogna stare attenti agli errori di traboccamento nella moltiplicazione. Ad esempio, la seguente espressione dà un traboccamento se il prodotto delle due variabili intere è al di fuori dell'intervallo permesso degli interi:

```
int1% * int2%
```

Un rimedio per questo errore potenziale è convertire uno degli operandi in un tipo con un intervallo più grande:

```
CLNG(int1%) * int2%
```

In questo caso, QuickBASIC converte entrambi gli operandi in interi lunghi prima di eseguire la moltiplicazione. (Vedere le voci CDBL, CLNG e CSNG per un approfondimento.) Nelle espressioni che eseguono operazioni composte, moltiplicazione e divisione hanno precedenza su addizione e sottrazione, a meno che la precedenza di default sia modificata dalle parentesi.

Un esempio

La seguente istruzione di assegnamento illustra la moltiplicazione, la conversione del tipo e l'utilizzo delle parentesi per modificare la precedenza di default delle operazioni:

```
num& = giorniPerAn& * (anno% - Annoiniz%)
```

Grazie alle parentesi, QuickBASIC esegue per prima la sottrazione. Il risultato della sottrazione viene poi convertito in un intero lungo, per corrispondere al tipo dell'altro operando, *giorniPerAn&*, e viene eseguita la moltiplicazione. (Questo esempio è preso dalla funzione *NumData&* nel programma *Mese*, presentato nel Capitolo 29.)

Divisione

L'operatore / rappresenta in QuickBASIC la divisione in virgola mobile:

dividendo / divisore

Gli operandi possono essere numeri in virgola mobile o interi. In entrambi i casi il risultato è sempre un numero in virgola mobile. La divisione per zero

non è definita in QuickBASIC e si ha un errore al momento dell'esecuzione. Nelle espressioni, moltiplicazione e divisione hanno precedenza su addizione e sottrazione, a meno che la precedenza di default sia modificata con le parentesi.

Un esempio

La seguente istruzione di assegnamento illustra la divisione, la conversione di tipo e l'utilizzo delle parentesi per modificare la precedenza di default delle operazioni:

```
CalcAnnImp = (AnnoCorr& - Iniz&) / 365
```

Grazie alle parentesi, QuickBASIC esegue per prima la sottrazione e poi la divisione. Anche se gli operandi sono interi, il risultato della divisione è un valore in virgola mobile, che viene assegnato a *CalcAnnImp*. (Questo esempio è preso dalla funzione *CalcAnnImp* nel programma *Compleanno*, presentato nel Capitolo 28.)

Divisione tra interi

L'operazione di divisione tra interi è rappresentata dal carattere backslash (\), nel seguente formato:

dividendo \ divisore

Il dividendo ed il divisore possono essere valori in virgola mobile o interi. Prima che venga eseguita la divisione, se necessario, entrambi gli operandi vengono arrotondati, con un'operazione che è equivalente all'azione della funzione incorporata CINT. Il risultato della divisione viene troncato. In QuickBASIC la divisione per zero non è definita e si ha un errore al momento dell'esecuzione. Poiché gli operandi vengono arrotondati nella divisione tra interi, il divisore deve essere maggiore di 0,5 o minore di -0,5.

Un esempio

La seguente istruzione di assegnamento fà parte di una routine di ricerca binaria chiamata *CercData*, del programma *Agenda* presentato nel Capitolo 30.

```
pos2% = (pos1% + pos3%) \ 2
```

Questa istruzione ha lo scopo di localizzare in una lista di valori la posizione posta a metà tra altre due. L'operazione di divisione tra interi dà un valore intero che è memorizzato in *pos2%*.

Operazione modulo

La parola chiave MOD consente di utilizzare l'aritmetica in modulo. Il risultato di questa operazione è il resto della divisione tra due interi:

dividendo MOD divisore

Il dividendo ed il divisore possono essere valori in virgola mobile o interi. Prima dell'esecuzione dell'operazione MOD, entrambi gli operandi vengono arrotondati con un'operazione equivalente all'azione della funzione incorporata CINT, se necessario. Il risultato di MOD è il resto intero della divisione. In QuickBASIC la divisione per zero non è definita e si ha un errore al momento dell'esecuzione. Poiché gli operandi nell'operazione MOD vengono arrotondati, il divisore deve essere maggiore di 0,5 o minore di -0,5.

Un esempio

Un utilizzo importante dell'operazione MOD è quello di determinare se un numero è divisibile per un altro. Se il risultato della divisione è un numero intero, l'operazione MOD dà zero. Ad esempio, le tre seguenti istruzioni hanno il compito di verificare se l'intero memorizzato nella variabile *anno%* rappresenta un anno bisestile:

```
divPer4% = (anno% MOD 4 = 0)
secolo% = (anno% MOD 100 = 0)
secolo400% = (anno% MOD 400 = 0)
```

Queste istruzioni assegnano risultati logici alle variabili *divPer4%*, *secolo%*, e *secolo400%*. In ogni caso, il risultato è vero se il divisore sta nel valore *anno%* ed il risultato dell'operazione MOD è zero.

Addizione e sottrazione

In QuickBASIC l'operatore + rappresenta l'addizione e - rappresenta la sottrazione.

```
val1 + val2    ' Addizione.
val1 - val2    ' Sottrazione.
```

Queste operazioni hanno precedenza minima rispetto a tutte le altre operazioni. Se si desidera che la sottrazione o l'addizione siano eseguite per prime, in un'espressione composta, bisogna inserirle tra parentesi. Si noti che in QuickBASIC l'operatore di addizione (+) viene usato anche per concatenare due stringhe.

Un esempio

La seguente espressione illustra l'uso delle parentesi per controllare l'ordine con cui vengono eseguite le operazioni:

```
(NumMese% - 1) * 10 + 1
```

Per valutare questa espressione, QuickBASIC esegue prima la sottrazione, poi la moltiplicazione ed infine l'addizione.

OPERAZIONI RELAZIONALI E LOGICHE

Questo paragrafo descrive sei operazioni relazionali e sei operazioni logiche. Queste operazioni sono essenziali per esprimere le condizioni logiche

che controllano i selettori e i cicli in un programma QuickBASIC. Ogni operazione relazionale confronta una coppia di valori, ne verifica l'uguaglianza o la disuguaglianza. Da parte loro, le operazioni logiche hanno il compito di confrontare coppie di valori logici per formare espressioni logiche composte. (L'unica eccezione è NOT, un'operazione unaria che capovolge il senso di un operando logico.) Ognuna di queste operazioni dà come risultato un valore logico vero o falso. In realtà, QuickBASIC non definisce un valore di tipo logico; al contrario, il risultato di un'operazione logica è registrato in memoria come un intero: il valore falso è rappresentato da 0 e quello vero da -1. Viceversa, QuickBASIC può leggere nell'opportuno contesto ogni intero come un valore logico e 0 viene considerato falso mentre ogni altro intero viene letto come vero. (I valori vero e falso talvolta vengono chiamati valori *booleani*, in onore di George Boole, un matematico inglese del XIX secolo.) Un'espressione logica complessa può contenere molte operazioni di vario tipo. In questo caso, l'ordine con cui le operazioni vengono valutate diventa un compito importante. QuickBASIC segue un ordine di default di precedenza nel valutare le operazioni. In generale, le operazioni aritmetiche vengono valutate per prime, poi vengono le operazioni relazionali ed infine quelle logiche. Vi è inoltre un ordine all'interno di ogni gruppo di operazioni. In particolare, le operazioni logiche vengono valutate secondo questa successione: NOT, AND, OR, XOR, EQV e IMP. Come nelle operazioni aritmetiche, si possono usare sempre le parentesi per escludere l'ordine di valutazione di default o semplicemente per motivi di chiarezza. Gli operatori logici talvolta vengono usati anche per manipolazioni di bit, ovvero della sequenza di cifre binarie che corrispondono ad un valore in memoria. In questo tipo di operazione, una cifra binaria di 0 equivale al valore logico falso ed una di 1 a vero. La prima voce in questo paragrafo descrive le operazioni relazionali come insieme. Poi ogni operazione logica viene trattata nella propria singola voce.

Operazioni relazionali

Le operazioni relazionali (rappresentate dai simboli =, <>, <, >, <= e >=) possono essere usate per confrontare coppie di operandi numerici. Ogni operazione dà come risultato un valore logico vero o falso.

```
val1 = val2    ' Uguaglianza.
val1 <> val2   ' Disuguaglianza.
val1 < val2    ' Minore di.
val1 > val2    ' Maggiore di.
val1 <= val2   ' Minore o uguale.
val1 >= val2   ' Maggiore o uguale.
```

Gli operandi del confronto possono essere valori numerici literal, variabili, chiamate di funzione che ritornano valori numerici o operazioni aritmetiche. Gli operandi possono appartenere ai tipi in virgola mobile o intero. In un'espressione che contiene operandi di tipi misti, QuickBASIC converte tutti i valori al tipo più preciso che compare nell'espressione. Le operazioni aritmetiche hanno la precedenza sulle operazioni relazionali. Per questo motivo, le parentesi sono facoltative in un'espressione che contiene entrambi i tipi di operazioni:

```
IF a% = b% - 1 THEN ...
```

Per valutare la condizione logica di questa istruzione IF, QuickBASIC esegue prima la sottrazione e poi determina se l'uguaglianza è vera o falsa. Si noti che in QuickBASIC le operazioni relazionali possono essere usate anche per confrontare le stringhe. (Vedere la voce "Confronto tra stringhe" per ulteriori dettagli.)

Alcuni esempi

Le istruzioni IF e i cicli DO spesso contengono condizioni logiche composte da una o più operazioni relazionali. Ad esempio, si consideri la seguente istruzione IF del programma *Agenda* (presentato nel Capitolo 30):

```
IF inVal% >= 1 AND inVal% <= 12 THEN Rev% = TRUE
```

Questa condizione è vera se *inVal%* contiene un intero tra 1 e 12, compresi. Si noti l'uso dell'operazione AND per unire le due espressioni relazionali. Ecco un altro esempio preso dal programma *Agenda*, questa volta da un'istruzione DO WHILE:

```
DO WHILE datTrov% = 0 AND pos1% <= pos3%
```

In questo caso il risultato dell'espressione condizionale determina se è il caso di iterare o no il ciclo. (Vedere le voci IF e DO nel Capitolo 7 per ulteriori informazioni.) Si può assegnare inoltre il risultato logico di un'espressione relazionale direttamente ad una variabile. Ecco un esempio da una routine di selezione nel programma *Compleanno* (Capitolo 28):

```
scamb% = staff(i%).etàAtt < staff(j%).etàAtt
```

Questa istruzione confronta un campo numerico (*etàAtt*) da due diversi record di un vettore chiamato *staff*. Il risultato logico del confronto, vero o falso, viene assegnato alla variabile intera *scamb%*. (Effettivamente, se l'espressione relazionale è falsa, QuickBASIC assegna il valore 0 a *scamb%*. Per un risultato vero, *scamb%* riceve il valore -1.) Il programma poi usa questo valore logico per decidere se effettuare o meno uno scambio:

```
IF scamb% THEN SWAP staff(i%), staff(j%)
```

Assegnare il risultato di un'espressione relazionale ad una variabile può diventare poco chiaro quando la relazione è l'operazione di uguaglianza (=). Si consideri il seguente esempio preso dal programma *Agenda*:

```
Corr% = (risp$ = "S")
```

Il fatto che questa istruzione contenga due segni di uguaglianza può far pensare che si stiano eseguendo due assegnamenti. (In altri linguaggi, il C ad esempio, starebbe accadendo proprio ciò.) Quello che sta succedendo è invece che il QuickBASIC valuta l'espressione relazionale *risp$ = "S"* come vera o falsa ed assegna il risultato di quest'espressione alla variabile intera *Corr%*.

Operazioni logiche

Le operazioni logiche rappresentate da NOT, AND, OR, XOR, EQV e IMP vengono applicate ai valori logici vero o falso per creare espressioni logiche composte.

NOT

L'operazione unaria NOT inverte il senso di un valore logico. L'espressione NOT A è vera se A è falso; la stessa espressione è falsa se A è vero.

```
NOT TRUE     ' Equivale a FALSE.
NOT FALSE    ' Equivale a TRUE.
```

Esempi Entrambi questi passaggi sono stralci del programma *Agenda*, discusso nel Capitolo 30. Il primo esempio è parte di una funzione che determina il numero di giorni in un dato mese. L'istruzione calcola come segue il valore della variabile intera *giorni%* se il mese è febbraio:

```
IF NumMese% = 2 AND NOT AnnoBisest%(anno%) THEN giorni% = giorni% - 1
```

Il valore di default del programma per *giorni%* nel mese di febbraio è 29. Se *NumMese%* è 2 (che corrisponde a febbraio) e l'intero memorizzato in *anno%* non è un anno bisestile, questa istruzione IF riduce il valore di *giorni%* a 1. Si noti che la seguente espressione è vera se la chiamata alla funzione *AnnoBisest%* ritorna un valore falso:

```
NOT AnnoBisest%(anno%)
```

Un secondo esempio di NOT compare nella seguente istruzione:

```
IF NOT dataValida% THEN anno% = 0
```

Qui il programma sta usando la variabile intera *dataValida%* come un valore logico. Questa istruzione assegna un valore zero all'*anno%* se *data Valida%* è falsa, cioè se la stessa variabile *dataValida%* contiene un valore pari a zero.

AND

L'operazione AND è vera solo se entrambi gli operandi logici sono veri. Se uno o entrambi gli operandi sono falsi, l'operazione AND dà un valore falso.

```
TRUE AND TRUE     ' Equivale a TRUE.
TRUE AND FALSE    ' Equivale a FALSE.
```

```
FALSE AND TRUE   ' Equivale a FALSE.
FALSE AND FALSE  ' Equivale a FALSE.
```

Esempi In ognuno di questi due esempi (entrambi estratti del programma *Agenda* presentato nel Capitolo 30) compare un'espressione AND come condizione di controllo per un ciclo DO. Il primo esempio legge, da tastiera, l'immissione di un carattere da parte dell'utente:

```
PRINT "Salva gli appuntamenti odierni? <S> o <N>";
DO
   risp$ = UCASE$(INKEY$)
LOOP UNTIL LEN(risp$) > 0 AND INSTR("SN", risp$) <> 0
```

La clausola UNTIL in questa struttura esamina il valore attuale di *risp$*. Per assicurare la presenza di un carattere d'immissione valido, il ciclo continua finché la lunghezza di *risp$* è maggiore di zero (cioè, quando l'utente ha effettivamente premuto un tasto) ed il tasto premuto è S o N. Una chiamata alla funzione incorporata INSTR determina se *risp$* è contenuta nella stringa "SN". Il secondo esempio, un ciclo DO WHILE, è parte di una routine di ricerca binaria che cerca un valore numerico con un indice selezionato:

```
DO WHILE dataTrov% = 0 AND pos1% <= pos3%
```

In questo caso, il ciclo continua fino a che non viene trovato il numero o la routine determina che il numero non è nella lista dell'indice. (Per esaminare questa routine più dettagliatamente, vedere la funzione *CercData%* nel programma *Agenda*, presentato nel Capitolo 30.)

OR

L'operazione OR è vera se uno o entrambi gli operandi logici sono veri. Se entrambi gli operandi sono falsi, l'operazione dà vero o falso.

```
TRUE OR TRUE    ' Equivale a TRUE.
TRUE OR FALSE   ' Equivale a TRUE.
FALSE OR TRUE   ' Equivale a TRUE.
FALSE OR FALSE  ' Equivale a FALSE.
```

Esempio Il seguente stralcio è preso da una funzione che si chiama *GiorniMese%*, il cui compito è determinare il numero di giorni di un dato

mese. Questo selettore IF usa un'operazione OR per determinare se il valore memorizzato nella variabile dell'argomento *numMese%* è valido:

```
IF numMese% < 1 OR numMese% > 12 THEN
   GiorniMese% = 0
   EXIT FUNCTION
END IF
```

La funzione si aspetta che *numMese%* contenga un intero nell'intervallo da 1 a 12. Se il valore non è compreso in questo intervallo, un'istruzione EXIT FUNCTION termina l'esecuzione della funzione.

XOR

L'operazione XOR (e non OR) è vera se esattamente uno degli operandi logici è vero. Se entrambi gli operandi sono veri o falsi, l'operazione XOR dà un valore falso.

```
TRUE XOR TRUE     ' Equivale a FALSE.
TRUE XOR FALSE    ' Equivale a TRUE.
FALSE XOR TRUE    ' Equivale a TRUE.
FALSE XOR FALSE   ' Equivale a FALSE.
```

Esempio Il seguente esempio potrebbe far parte di un programma di giochi che richiede al giocatore di muovere un segnalino da una posizione all'altra sullo schermo. Le regole del gioco specificano che l'utente (1) deve spostarsi dalla posizione di partenza e (2) muoversi orizzontalmente o verticalmente, ma non in diagonale. In questa parte viene usata l'operazione XOR in un selettore IF per determinare se il giocatore ha eseguito un movimento corretto:

```
IF stOrizz% XOR stVertic% THEN
   buonMos% = TRUE
   LOCATE 2, 50
   PRINT "Mossa corretta: ";
   IF stOrizz% THEN PRINT "Vertic." ELSE PRINT "Orizz."
ELSE
   buonMos% = FALSE
   LOCATE 23, 10
   PRINT "Errore: ";
   IF stOrizz% AND stVertic% THEN
```

```
      PRINT "Devi spostarti in un'altra posizione."
   ELSE
      PRINT "Gli spostamenti diagonali non sono ammessi."
   END IF
END IF
```

Il programma usa le variabili *stOrizz%* e *stVertic%* come valori logici, indicando se la nuova posizione ha le stesse coordinate orizzontali e/o verticali della posizione precedente. Se l'operazione XOR è vera, una delle coordinate è cambiata ed il giocatore ha seguito le regole del gioco esattamente. (Vedere la voce EQV per un'altra versione di questo esempio.)

EQV

EQV rappresenta l'operazione di equivalenza. L'espressione logica A EQV B è vera se A e B hanno lo stesso identico valore logico, cioè se entrambe sono o vere o false. Se A e B hanno diversi valori logici, l'espressione è falsa.

```
TRUE EQV TRUE     ' Equivale a TRUE.
TRUE EQV FALSE    ' Equivale a FALSE.
FALSE EQV TRUE    ' Equivale a FALSE.
FALSE EQV FALSE   ' Equivale a TRUE.
```

Esempio Il seguente frammento di codice è una versione alternativa dell'esempio presentato nella voce XOR. Un programma di giochi richiede che il giocatore muova un segnalino dello schermo in una nuova posizione che ha esattamente una coordinata in comune con la posizione attuale. (Il giocatore può muoversi orizzontalmente o verticalmente, ma non in diagonale.) In questa versione del programma viene usata l'operazione EQV in un selettore IF per determinare se il giocatore ha eseguito una mossa possibile:

```
IF stOrizz% EQV stVertic% THEN
   buonMos% = FALSE
   LOCATE 23, 10
   PRINT "Errore: ";
   IF stOrizz% AND stVertic% THEN
      PRINT "Devi spostarti in un'altra posizione."
   ELSE
      PRINT "Gli spostamenti diagonali non sono ammessi."
   END IF
```

```
ELSE
   buonMos% = TRUE
   LOCATE 2, 50
   PRINT "Mossa permessa: ";
   IF stOrizz% THEN PRINT "Vertic." ELSE PRINT "Orizz."
END IF
```

Le variabili logiche *stOrizz%* e *stVertic%* indicano se la nuova posizione ha le stesse coordinate orizzontali e/o verticali della precedente posizione. In questo caso la mossa non è ammessa se l'operazione EQV è vera: il giocatore non ha effettuato nessuno spostamento (entrambi i valori logici sono veri) o ne ha fatto uno in diagonale (entrambi i valori logici sono falsi). Si noti che EQV è l'inverso di XOR: per ogni combinazione degli operandi logici, il risultato dell'operazione EQV è l'opposto di XOR.

IMP

IMP rappresenta l'operazione di implicazione. L'espressione logica A IMP B è falsa se A è vero e B è falso; in tutti gli altri casi l'espressione è vera.

```
TRUE IMP TRUE     ' Equivale a TRUE.
TRUE IMP FALSE    ' Equivale a FALSE.
FALSE IMP TRUE    ' Equivale a TRUE.
FALSE IMP FALSE   ' Equivale a TRUE.
```

Esempio La seguente funzione, chiamata *AnnoBisest%*, riceve un singolo argomento intero che rappresenta un anno del calendario e ritorna un valore logico vero se l'argomento è un anno bisestile o falso se non lo è:

```
FUNCTION AnnoBisest% (anno%)

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)
   AnnoBisest% = divPer4% AND (secolo% IMP secolo400%)

END FUNCTION
```

Gli anni che sono perfettamente divisibili per 4 sono anni bisestili, con questa eccezione: un secolo (1800, 1900, 2000 e così via) è un anno bisestile solo se è perfettamente divisibile per 400. Di conseguenza, la seguente

espressione dà un valore vero in tutti i casi ad eccezione degli anni che sono perfettamente divisibili per 100 e non per 400:

```
secolo% IMP secolo400%
```

Per esaminare la routine *AnnoBisest%* nel contesto di un'applicazione, vedere il programma *Compleanno* (Capitolo 28), il programma *Mese* (Capitolo 29) o il programma *Agenda* (Capitolo 30).

Capitolo 2

Dichiarazioni

Questo capitolo descrive i molti tipi di dichiarazione che si possono utilizzare per predefinire il tipo e la visibilità delle variabili, per rappresentare i dati nel modo più chiaro ed opportuno, ed infine per rendere i propri programmi più facili da scrivere, leggere e correggere.

COSTANTI SIMBOLICHE E TIPI DI VARIABILI

L'istruzione CONST, che dichiara le costanti simboliche, è il primo argomento del capitolo. Una costante simbolica è un nome attribuito ad un valore costante in un programma QuickBASIC. Si possono usare costanti simboliche per rappresentare dati che assumono valori fissi, sia che si tratti di numeri sia di stringhe, e che sono essenziali per l'esecuzione di un programma. La seconda voce di questo capitolo riguarda le istruzioni DEF*tipo* del QuickBASIC. Il metodo tradizionale per dichiarare il tipo delle variabili in BASIC è quello di includere uno specifico carattere di tipo alla fine del nome della variabile. Se, però, inavvertitamente si è omesso il carattere che specifica il tipo dal nome della variabile esistente, questo approccio può far sì che ci si confonda. Come approccio alternativo, QuickBASIC offre un gruppo di istruzioni DEF*tipo*, che prestabiliscono alcune lettere dell'alfabeto per rappresentare i tipi di dati specificati.

CONST

Una dichiarazione CONST crea uno o più *costanti simboliche* da utilizzare in un programma.

```
CONST nome = val, ...     ' Possono essere definiti più nomi.
```

Una costante simbolica è un nome usato per rappresentare un particolare numero o stringa in un programma. A differenza di una variabile, una costante simbolica rappresenta un valore invariabile nel corso dell'esecuzione del programma. Una volta dichiarata una costante simbolica, si può usare il suo nome in ogni espressione del QuickBASIC, proprio come si utilizzerebbe il nome di una variabile. L'unica restrizione è che il programma non può modificare il valore di una costante simbolica. Il tentativo di comportarsi in questo modo risulta nel messaggio d'errore al momento dell'esecuzione "Definizione doppia". Il nome scelto per una costante simbolica può iniziare con una lettera dell'alfabeto e può comprendere fino a 40 lettere o cifre. (È ammesso anche il punto.) Nella dichiarazione CONST si può includere uno dei suffissi di tipo del QuickBASIC per dichiarare il tipo del valore che la costante assumerà: %, &, !, # o $. Se si omette questo simbolo, QuickBASIC determina il tipo di dati dal valore che si specifica per la costante simbolica. In ogni caso, il suffisso del tipo è sempre facoltativo negli utilizzi successivi della costante simbolica. Nei nomi delle costanti simboliche non ha importanza che le lettere siano scritte in maiuscolo o minuscolo. Alcuni programmatori amano usare tutte le lettere maiuscole, per distinguere le costanti dai nomi delle variabili. Nell'istruzione CONST solitamente si esprime il valore di una costante simbolica come valore numerico literal o stringa literal tra virgolette. Si può anche specificare il valore attraverso un'espressione che può comprendere qualsiasi operazione aritmetica o logica (eccetto l'elevamento a potenza) e contiene valori literal o altre costanti simboliche come operandi. Una singola istruzione CONST può dichiarare più costanti simboliche oppure il programma può contenere una lista di istruzioni CONST per definire singole costanti. Le costanti simboliche dichiarate nella porzione principale del programma sono globali, cioè sono utilizzabili in tutte le procedure del programma. È possibile anche dichiarare costanti simboliche per uso locale all'interno di una procedura o di una funzione.

Alcuni esempi della dichiarazione CONST

Il programma *Compleanno*, presentato nel Capitolo 28, stabilisce le seguenti costanti simboliche per rappresentare diversi ordini disponibili per visualizzare un database degli impiegati:

```
CONST ORDINENOM = 1     ' Ordina secondo il nome degli impiegati.
CONST ORDINEQUAL = 2    ' Ordina secondo la qualifica.
CONST ORDINEANZA = 3    ' Ordina secondo gli anni di anzianità,
                        ' ascendente.
CONST ORDINEANZD = 4    ' Ordina secondo gli anni di anzianità,
                        ' discendente.
CONST ORDINEETA = 5     ' Ordina secondo l'età.
CONST ORDINECOMP = 6    ' Ordina secondo i compleanni.
```

La routine di ordinamento del programma usa questi nomi in una struttura di decisione SELECT CASE per scegliere un campo chiave per l'ordinamento:

```
SELECT CASE Chiave%

   CASE ORDINENOM
      Nomin1$ = staff(i%).Cogn + staff(i%).Nome
      Nomin2$ = staff(j%).Cogn + staff(j%).Nome
      scamb% = Nomin1$ > Nomin2$

   CASE ORDINEQUAL
      scamb% = staff(i%).Anzian > staff(j%).Anzian

   ' ...
```

Le costanti simboliche sono convenienti anche per rappresentare speciali caratteri ASCII usati in un programma. Ad esempio, la dichiarazione seguente compare nel programma *Agenda* (Capitolo 30), in un sottoprogramma chiamato *LeggiIndData*:

```
CONST slash = "/"
```

La routine usa *slash* con il compito di cercare i caratteri separatori in una stringa di dati:

```
posSlash1% = INSTR(inData$, slash)
posSlash2% = INSTR(posSlash1% + 1, inData$, slash)
```

Infine, molti programmi di questo libro indicano costanti simboliche con il nome *TRUE* e *FALSE* per rappresentare valori logici:

```
CONST FALSE = 0
CONST TRUE = NOT FALSE
```

Nelle espressioni logiche, QuickBASIC legge il valore zero come falso e tutti i valori diversi da zero come veri. Si noti che la seconda di queste due dichiarazioni CONST dichiara il valore TRUE come l'espressione NOT FALSE. (Per ulteriori informazioni consultare il paragrafo sulle operazioni logiche nel Capitolo 1.)

Compatibilità

Il Turbo Basic non ha la dichiarazione CONST, ma consente una speciale notazione per dichiarare le *costanti con nome*: come primo carattere del nome si ha il simbolo di percentuale (%). Ad esempio, l'istruzione seguente crea nel Turbo Basic la costante nominata %NAMESORT:

```
%NAMESORT = 1
```

In Turbo Basic le costanti con nome possono rappresentare solo valori interi. BASICA non ammette l'uso di costanti simboliche.

DEF*tipo*

Le dichiarazioni DEF*tipo* permettono di predichiarare i tipi delle variabili e delle funzioni che iniziano con lettere dell'alfabeto selezionate.

```
DEFINT lettere   ' Dichiara variabili intere.
DEFLNG lettere   ' Dichiara variabili intere-lunghe.
DEFSNG lettere   ' Dichiara variabili in precisione semplice.
DEFDBL lettere   ' Dichiara variabili in doppia precisione.
DEFSTR lettere   ' Dichiara variabili stringa.
```

Le cinque istruzioni DEF consentono una comoda alternativa all'uso dei suffissi (%, &, !, # e $) per dichiarare i tipi di dati di variabili e di funzioni.

Un'istruzione DEF*tipo* specifica una lista di singole lettere o un serie di lettere. Ogni elemento nella lista di lettere può essere sia una singola lettera dell'alfabeto, come:

```
DEFINT I
```

o un intervallo di lettere:

```
DEFINT M-P
```

Si possono anche includere combinazioni di singole lettere e intervalli:

```
DEFINT I, K, M-P, R
```

Nella lista non è importante che le lettere siano scritte maiuscole o minuscole. (Infatti, l'editor del QuickBASIC in un'istruzione DEF*tipo* converte automaticamente le lettere minuscole in maiuscole.) Ogni nome di variabile che inizia con una delle lettere comprese nella lista apparterrà, per default, al tipo di dati specificato. Una dichiarazione DEF*tipo* si applica anche ai nomi di funzione definiti in una struttura DEF FN o in una procedura FUNCTION. Per escludere il tipo DEF dichiarato, si può sempre includere un suffisso di tipo alla fine di un nome di variabile o di funzione. L'editor del QuickBASIC automaticamente copia ogni dichiarazione DEF*tipo* all'inizio delle procedure in cui la dichiarazione è attinente. Ad esempio, si supponga di inserire la seguente dichiarazione nella parte iniziale di un programma:

```
DEFSTR T
```

Poi si scrive una nuova procedura di funzione chiamata *Titolo*. QuickBASIC automaticamente copia la dichiarazione DEFSTR nella riga situata proprio sopra la nuova istruzione FUNCTION:

```
DEFSTR T
FUNCTION Titolo
   ' ...
```

*Alcuni esempi di istruzioni di DEF*tipo

Il seguente breve programma, che dimostra le conversioni di dati numerici, contiene quattro istruzioni DEF*tipo* per dichiarare i tipi delle variabili usate nel programma:

```
DEFLNG L
DEFSNG S
DEFINT I
DEFDBL D

' dimostrazione dell'uso di CDBL.

valLung1 = 778655989
valLung2 = 113466691

valDopp = CDBL(valLung1) * valLung2
PRINT valDopp

' dimostrazione dell'uso di CLNG.

valInt1 = 1234
valInt2 = 5678

valLung = CLNG(valInt1) * valInt2
PRINT valLung

' dimostrazione dell'uso di CSNG.

valInt1 = 1234
valInt2 = 5678

valSing = CSNG(valInt1) * valInt2
PRINT valSing
```

Grazie alle quattro istruzioni DEF*tipo* all'inizio del listato, questo programma non deve usare il suffisso di tipo (%, & e #) per dichiarare i diversi tipi delle variabili numeriche. Ad esempio, la variabile *valLung* appartiene per default al tipo di dati interi lunghi, grazie a questa istruzione:

```
DEFLNG L
```

Si possono avere ulteriori dettagli riguardo a questo programma consultando le voci CDBL, CLNG e CSNG.

Compatibilità

Il Turbo Basic supporta le stesse istruzioni DEF*tipo* come il QuickBASIC. BASICA ha solo la dichiarazione DEFLNG; gli interi lunghi non sono definiti in BASICA.

VISIBILITÀ

La *visibilità* di una variabile è la porzione di codice in cui il nome di una variabile viene riconosciuto nei diversi moduli e procedure di un programma. Solitamente, una variabile *locale* è definita solo per l'uso all'interno di un dato sottoprogramma o funzione, mentre una variabile *globale* è utilizzabile in qualsiasi punto di un programma. Le opzioni di visibilità sono tra le più importanti innovazioni del compilatore di QuickBASIC: infatti nelle versioni interpretate del BASIC tutte le variabili sono globali, il che risulta essere un grosso limite nei lunghi progetti di programmazione. QuickBASIC ha tre dichiarazioni che definiscono la visibilità di una variabile: COMMON, SHARED e STATIC. Si può usare la prima di queste, COMMON, per condividere le variabili tra tutte le procedure di un modulo di un dato programma o per scambiare i dati tra le parti di un programma che consiste in più moduli. (La voce COMMON di questo paragrafo spiega l'utilizzo dei moduli in QuickBASIC.) In alternativa, la dichiarazione SHARED consente ad un programma di riconoscere specifiche variabili come utilizzate contemporaneamente in una specifica procedura e nella parte principale del programma. Infine, la dichiarazione STATIC stabilisce la visibilità di una variabile come legata ad una particolare procedura. La clausola AS in queste tre istruzioni consente di dichiarare il tipo di dati (o struttura) di una variabile nello stesso momento in cui si definisce il campo di validità. Si possono avere altri particolari su queste tre istruzioni, leggendo le pagine seguenti di questo capitolo.

COMMON

Si può usare l'istruzione COMMON in tre diversi contesti: 1) per rendere alcune variabili utilizzabili in tutte le procedure di un modulo dato; 2) per

condividere dati tra i moduli; 3) per trasferire i dati in un programma *concatenato*.

```
COMMON SHARED var     ' Per dichiarare le var globali all'interno
                      ' di un modulo.
```

o

```
COMMON SHARED /NomeBloc/ var     ' Per condividere i dati tra i moduli.
                                 ' SHARED e /NomeBloc/
                                 ' sono facoltativi.
```

o

```
COMMON SHARED var     ' Per trasferire i dati in un
                      ' programma concatenato.
                      ' SHARED è facoltativo.
```

In tutti e tre questi contesti, l'istruzione COMMON compare sempre nella parte principale del programma (o codice "livello del modulo") precedendo la prima istruzione eseguibile. La lista delle variabili nell'istruzione COMMON può comprendere sia nomi di variabili scalari sia nomi di vettori. Si può specificare il tipo di ogni variabile includendo un carattere di tipo ($, %, &, ! o #) alla fine del nome. In alternativa, si può includere una clausola AS, secondo questo formato:

```
var AS tipo
```

dove *tipo* è una delle parole chiave di QuickBASIC STRING, INTEGER, LONG, SINGLE o DOUBLE, o il nome di un tipo definito dall'utente. Il nome di un vettore nella lista COMMON deve essere seguito da due parentesi

```
vett()
```

L'istruzione DIM per un vettore *statico* precede sempre l'istruzione COMMON. Lo spazio per un vettore dinamico si trova invece più avanti nel programma, e comunque dopo l'istruzione COMMON. (Per ulteriori dettagli consultare la voce DIM . Si tenga presente che una dichiarazione DIM per un vettore dinamico è un'istruzione eseguibile e perciò deve comparire dopo la dichiarazione COMMON.) Le parti seguenti descrivono i vari compiti dell'istruzione COMMON.

Dichiarazione di variabili globali in un modulo

Un'istruzione COMMON SHARED dichiara una lista di variabili e vettori da utilizzare globalmente in un dato modulo. Si può usare questa istruzione per condividere i valori di alcune variabili tra tutte le procedure. In questo senso, COMMON SHARED è un'alternativa al trasferimento dei valori come argomenti nelle singole procedure. Ad esempio, le strutture di dati principali di un programma, cioè quelle usate da molti o dalla maggior parte dei sottoprogrammi e funzioni nel programma, potrebbero opportunamente essere rappresentate come variabili globali. Comunque, si tenga presente che la potenza di QuickBASIC sta in parte nel suo supporto per procedure strutturate, con parametri e variabili locali. Progettare programmi che si basano eccessivamente sulle variabili globali vuol dire ritornare alle regole inefficienti e poco chiare di programmazione del BASIC interpretato. (Consultare le voci SUB e FUNCTION per ulteriori dettagli. Si osservi che l'istruzione DIM SHARED può essere usata per dichiarare variabili e vettori globali all'interno di un dato modulo. Per altri particolari a riguardo, consultare la voce DIM.)

Passaggio di dati tra moduli

È possibile utilizzare l'istruzione COMMON per condividere i dati tra diversi *moduli*. I seguenti paragrafi descrivono l'uso dei moduli in QuickBASIC. L'ambiente di programmazione QuickBASIC consente di caricare in memoria contemporaneamente più programmi. Questi diventano i *moduli* del proprio progetto di programmazione. Ogni modulo potenzialmente è composto da un programma principale (chiamato anche codice del "livello del modulo", cioè il codice che non fa parte di nessuna procedura) ed una raccolta di procedure. Il programma principale del *modulo principale* controlla l'azione dell'intero programma, tramite chiamate alle procedure che si trovano in uno dei moduli presenti in memoria. Quando si salva un programma che è formato da più moduli, ogni modulo viene scritto separatamente nel proprio file su disco. QuickBASIC crea anche uno speciale file di testo, con estensione MAK, che contiene una lista di tutti i nomi dei moduli presenti nel programma. Nel processo di compilazione un programma a più moduli su disco, QuickBASIC crea un solo file EXE da

tutti i file di codice sorgenti. Ecco i comandi principali del menù che sono d'aiuto per lavorare con i moduli nell'ambiente QuickBASIC:

- Il comando Open Programm del menù File apre tutti i moduli di un programma a più moduli se esiste un file MAK su disco per il programma specificato.
- Il comando Save All del menù File salva tutti i moduli di un programma su disco e crea un file MAK per il programma.
- Il comando Create File del menù File crea un modulo supplementare e lo include nel programma corrente.
- Il comando Load File del menù File carica un modulo aggiuntivo dal disco e lo include nel programma corrente.
- Il comando Unload File del menù File rimuove un modulo dal programma corrente.
- I comandi New Sub e New Function del menù View creano nuove procedure in un dato modulo.
- Il comando Sub del menù View presenta una lista di tutti i moduli e delle procedure del programma corrente. Inoltre, esso permette di selezionare un modulo o una procedura per l'editing. È possibile anche utilizzare questo comando per cancellare una procedura da un modulo o spostare una procedura da un modulo all'altro.
- Il File eseguibile Make del menù Run consente di stabilire un modulo di controllo per un programma a più moduli.

Ogni parte di un programma può chiamare qualsiasi procedura e trasferire con successo i valori nei suoi argomenti, anche se la procedura si trova in un modulo separato. Comunque, talvolta si può scegliere di condividere i dati a livello dei programmi principali di diversi moduli: l'istruzione COMMON consente di farlo. Ogni modulo che condivide i dati in questo modo deve avere la propria istruzione COMMON, posizionata vicino o sopra il codice a livello del modulo. (Un'istruzione COMMON non compare mai all'interno di una procedura.) Chiaramente le istruzioni COMMON corrispondenti in due diversi moduli non devono contenere gli stessi nomi di variabili. Piuttosto, i valori dei dati vengono inviati da un modulo all'altro tramite le *posizioni* dei nomi delle variabili nelle due liste COMMON. Ad esempio, supponiamo che un modulo abbia questa lista:

```
COMMON ditt$, imp%, entrate
```

Un altro modulo potrebbe rappresentare questi tre valori di dati condivisi attraverso tre nomi di variabili completamente diversi:

```
COMMON denAff$, dimstaff%, guad
```

L'unica richiesta è che le variabili corrispondenti, nelle due istruzioni COMMON, appartengano agli stessi tipi di dati. Nell'esempio, il valore memorizzato nella variabile di stringa *ditt$* viene trasferito nella variabile *denAff$* nel secondo modulo. Allo stesso modo, i valori numerici memorizzati in *imp%* e in *entrate* vengono trasferiti in *dimstaff%* e in *guad*. Un'istruzione COMMON può anche avere un *nome di blocco* che identifica il gruppo di variabili listate nell'istruzione. Il nome del blocco inizia con una lettera dell'alfabeto ed è composto al massimo da 40 lettere e cifre. Il nome è tra slash e compare prima della lista di variabili di una particolare istruzione COMMON:

```
COMMON /nomeBloc/ var1, var2, ...
```

Un blocco comune con nome può essere condiviso unitariamente con un particolare modulo del proprio programma. Un modulo che condivide con un altro un blocco con nome deve contenere un'istruzione COMMON con lo stesso nome del blocco. (Si troverà una breve trattazione dei blocchi condivisi con nome nei paragrafi "Esempi" di questo capitolo.)

Trasferimento di dati in un programma concatenato

Il comando CHAIN è un modo di terminare un programma ed iniziare l'esecuzione di un secondo programma. La concatenazione è essenzialmente una tecnica di stile BASICA, antiquata rispetto alla nuova programmazione modulare di QuickBASIC. Si potrebbe, comunque, aver bisogno di utilizzare il comando CHAIN in un vasto progetto di programmazione quando non vi è sufficiente memoria utilizzabile per organizzare il programma in qualsiasi altro modo. L'istruzione COMMON permette di trasferire valori di dati dal programma corrente ad un programma concatenato. Un'istruzione COMMON deve comparire nel programma

principale di entrambi i programmi. I valori vengono trasferiti sulla base della posizione dei nomi delle variabili nelle due liste COMMON. Le variabili corrispondenti nelle relative liste devono appartenere allo stesso tipo di dati. Il nome di un blocco non può essere usato nella concatenazione.

Alcuni esempi di istruzioni COMMON

Il programma *Compleanno*, presentato nel Capitolo 28, contiene un'istruzione COMMON SHARED che dichiara una variabile globale:

```
COMMON SHARED dimStaff%
```

La variabile *dimStaff%* rappresenta il numero di record nell'archivio impiegati del programma. Queste informazioni vengono richieste nel programma da molte procedure e possono essere gestite più opportunamente come una variabile globale. Allo stesso modo, il programma *Agenda* (presentato nel Capitolo 30) utilizza un'istruzione COMMON per dichiarare una variabile globale ed un vettore globale:

```
COMMON SHARED contImm%, IndData() AS immInd
```

La variabile *contImm%* rappresenta il numero di record attualmente memorizzati nell'archivio del calendario del programma ed il vettore *IndData* è un indice con cui il programma lavora, in molte differenti procedure, per localizzare specifici record nell'archivio. Per questo scopo, *IndData* è un vettore dinamico, prima dimensionato in una procedura con il nome *AprIndData*. Il seguente breve programma ha il compito di mostrare l'utilizzo di COMMON nei programmi multimoduli. Il programma è formato da tre moduli, con i nomi SOCIETA.BAS, GENERAL.BAS e PRODOTTO.BAS:

```
' SOCIETA.BAS
' Indica come usare la dichiarazione COMMON per la condivisione dati
' tra i diversi moduli di un programma.

DECLARE SUB InfGen ()
DECLARE SUB InfProd ()
```

```
COMMON /aGen/ ditt$, imp%, entr
COMMON /aProd/ prod$, unit&

ditt$ = "XYZ S.p.a."
imp% = 2154
entr = 3.000.000.000
prod$ = "Materiali Plastici"
unit& = 22000

CLS
PRINT "Profilo Ditta"
PRINT "——— ——"

InfGen
InfProd

END


' GENERAL.BAS
' Visualizza informazioni generali sulla ditta.

COMMON SHARED /aGen/ denAff$, dimStaff%, guad
SUB InfGen

   PRINT "Nome ditta:     "; denAff$
   PRINT USING "Numero di impiegati: ##.###"; dimStaff%
   PRINT USING "Guadagni annuali: L.##.###.####; guad
   PRINT

END SUB


' PRODOTTO.BAS
' Visualizza informazioni sul prodotto principale della ditta.

COMMON SHARED /aProd/ nomeArt$, artPerAnno&

SUB ProductInfo

   PRINT "Principale prodotto:  "; nomeArt$
   PRINT USING "Unità all'anno: ##,#######"; artperAnno&

END SUB
```

Come si può vedere, il modulo principale, *Ditta*, contiene un blocco comune denominato *aGeneral* per inviare i dati al modulo *General* ed un altro blocco comune *aProdotto* per inviare i dati al modulo *Prodotto*. I blocchi comuni

nei due moduli secondari usano diversi nomi di variabili, ma le variabili corrispondenti appartengono agli stessi tipi di dati. Il modulo principale inizializza le variabili e le procedure nei moduli secondari visualizzano i dati sullo schermo. Si noti che le istruzioni nei moduli *General* e *Prodotto* contengono la parola chiave SHARED per rendere disponibili i dati alle loro rispettive procedure.

Compatibilità

Le istruzioni COMMON in Turbo Basic e in BASICA sono disponibili solo per la concatenazione. Nessuna delle versioni del linguaggio supporta i blocchi con nome né l'attributo SHARED in COMMON. (In Turbo Basic, una procedura può utilizzare un'*istruzione* SHARED per ottenere l'accesso alle variabili create a livello del programma principale. In BASICA tutte le variabili sono globali.)

SHARED

L'istruzione SHARED appare solo all'interno di un sottoprogramma (SUB) o di una funzione (FUNCTION). Essa dichiara una lista di variabili e/o vettori che sono *condivisi*. Una variabile condivisa è utilizzabile sia nella sezione del programma principale, sia nella procedura in cui appare l'istruzione SHARED.

```
SHARED var AS tip, ...        ' La clausola AS è facoltativa.
```

In generale, il compito di SHARED è consentire ad una procedura di utilizzare una variabile che è già stata dichiarata nel programma principale. La lista di variabili dell'istruzione SHARED può comprendere nomi di variabili scalari, di vettori seguiti da parentesi vuote e record. La dichiarazione di ogni nome della lista deve essere identica alla sua dichiarazione originale nel programma principale. Se il tipo di variabile o vettore è dichiarato con una clausola AS nella parte del programma principale, la stessa clausola AS viene richiesta nell'istruzione SHARED. Le parole chiave per esprimere il tipo sono STRING, INTEGER, LONG, SINGLE e DOUBLE. Inoltre, la clausola AS può specificare un tipo definito dall'uten-

te. Si noti che l'istruzione COMMON SHARED, che può esistere solo nel programma principale, dichiara che una lista di variabili sono globalmente utilizzabili da tutte le procedure del programma. Le variabili COMMON SHARED sono utilizzabili per tutto il modulo; al contrario, le variabili SHARED lo sono solo nel programma principale e nella procedura che contiene l'istruzione SHARED. (Vedere la voce COMMON per ulteriori informazioni.)

Un esempio di istruzione SHARED

Il programma *Compleanno*, presentato nel Capitolo 28, usa diverse variabili globali e vettori. In particolare, la maggior struttura di dati del programma è un vettore globale di record chiamato *staff*: questo vettore è progettato per caricare in memoria l'archivio degli impiegati durante l'esecuzione del programma. Il vettore viene dichiarato nell'istruzione seguente:

```
DIM SHARED staff(dimStaff%) AS imp
```

Il tipo di record *impiegati* viene dichiarato precedentemente nel programma con un'istruzione TYPE. Grazie all'attributo SHARED in questa istruzione DIM, il vettore *staff* è utilizzabile da tutte le procedure del programma. Un metodo alternativo per controllare il campo di validità del vettore *staff* potrebbe essere utilizzare le istruzioni SHARED solo in quelle procedure che devono effettivamente usare il vettore. Con questa disposizione, l'istruzione DIM nella parte del programma principale si presenterebbe così:

```
DIM staff(dimStaff%) AS imp
```

Poi ogni procedura che utilizza il vettore *staff* inizierebbe con un'istruzione SHARED; ad esempio:

```
SUB LeggiRecStaff
   SHARED staff() AS imp
   ' ...
```

Le parentesi vuote dopo *staff* specificano che il nome rappresenta un vettore; la clausola AS corrisponde alla dichiarazione originale del vettore nella parte principale del programma. La scelta tra un'istruzione COMMON

SHARED nel programma principale e una sequenza di istruzioni SHARED all'inizio di procedure selezionate è generalmente un questione di preferenze personali. L'istruzione SHARED talvolta può essere stilisticamente superiore a COMMON SHARED, ma solo se SHARED documenta la lista di variabili che una data procedura condivide con il programma principale.

Compatibilità

Turbo Basic comprende l'istruzione SHARED, ma non supporta la clausola SHARED nelle istruzioni DIM e COMMON. Per questo motivo, SHARED è l'unico modo per condividere le variabili nel Turbo Basic. In BASICA tutte le variabili sono globali; ciò è uno dei problemi più noti del BASIC interpretato.

STATIC

L'istruzione STATIC compare solo all'interno di un sottoprogramma (SUB) o di una funzione (FUNCTION). Dichiara che una lista di variabili e/o vettori hanno visibilità locale, cioè sono utilizzabili solo nella procedura che contiene l'istruzione STATIC. Una variabile locale mantiene il proprio valore tra successive chiamate alla procedura.

```
STATIC var AS tip, ...        ' La clausola AS è facoltativa.
```

In generale, il fine di STATIC è distinguere le variabili locali da quelle con lo stesso nome che si trovano al di fuori di una procedura data. Inoltre, l'istruzione STATIC consente alle variabili di conservare il loro valore tra le chiamate ad una procedura. (Comunque, si può implementare questa seconda caratteristica più direttamente includendo la parola chiave STATIC nelle istruzioni SUB o FUNCTION all'inizio di una procedura.) La lista di variabili dell'istruzione STATIC può comprendere nomi di variabili scalari, di vettori seguiti da parentesi vuote e di record. Si può specificare il tipo di ogni variabile includendo un carattere di tipo ($,%,&,! o #) alla fine del nome della variabile. In alternativa è possibile includere una clausola AS con una delle parole chiave di tipo (STRING, INTEGER, LONG, SINGLE e DOUBLE). Inoltre, una clausola AS può specificare un tipo di record definito dall'utente.

Un esempio di istruzione STATIC

Il seguente breve programma di prova dimostra la differenza tra l'istruzione STATIC stessa e la parola chiave STATIC nell'istruzione SUB:

```
' STATIC test.

DECLARE SUB Test1 ()
DECLARE SUB Test2 ()

COMMON SHARED i%
i% = 5

CLS
PRINT "Test1: ";
FOR t% = 1 TO 5
   Test1
NEXT t%
PRINT

PRINT "Test2: ";
FOR t% = 1 TO 5
   Test2
NEXT t%

PRINT
PRINT USING "Dopo: ####"; i%

END

SUB Test1 STATIC
   i% = i% + 1
   PRINT USING "####"; i%;
END SUB

SUB Test2
   STATIC i%
   i% = i% + 1
   PRINT USING "####"; i%;
END SUB
```

Il programma inizia fissando la variabile *i%* come variabile globale ed assegnando alla variabile un valore iniziale di 5:

```
COMMON SHARED i%
i% = 5
```

I due cicli FOR effettuano una serie di chiamate ai due sottoprogrammi, *Test1* e *Test2*. Ognuna di queste due routine visualizza una serie di valori *i%* sullo schermo. Ogni chiamata a *Test1* incrementa la variabile globale *i%* di 1 e visualizza il nuovo valore sul video:

```
SUB Test1 STATIC
   i% = i% + 1
   PRINT USING "####"; i%;
END SUB
```

La parola chiave nell'istruzione SUB di *Test1* non impedisce alla procedura di riconoscere la variabile *i%* come globale, come si può vedere dall'output risultante:

```
Test1:   6 7 8 9 10
```

Test2 usa invece *i%* come una variabile locale:

```
SUB Test2
   STATIC i%
   i% = i% + 1
   PRINT USING "####"; i%;
END SUB
```

La manipolazione di *i%* in *Test2* non ha nessuna relazione con quella della variabile globale con lo stesso nome, come si può vedere dall'emissione ottenuta da *Test2*:

```
Test2:   1 2 3 4 5
```

Dopo l'ultima chiamata a *Test2*, il programma principale visualizza il valore finale della variabile globale *i%*:

```
PRINT USING "Dopo: ####"; i%
```

Il valore di questa variabile non è stato cambiato dalle chiamate a *Test2*:

```
Dopo: 10
```

Concludendo, l'istruzione STATIC crea una variabile locale che è diversa da ogni altra variabile usata al di fuori della procedura.

Compatibilità

Turbo Basic supporta l'istruzione STATIC, ma non l'attributo STATIC nelle istruzioni SUB. In BASICA tutte le variabili sono globali e STATIC non è supportata.

Capitolo 3

Vettori e record

Una struttura dati fa corrispondere valori multipli di dati ad un solo nome di variabile. Questo capitolo descrive le strutture dati disponibili in QuickBASIC. Un *vettore* è una struttura che può rappresentare una lista o una tabella di valori. Ogni voce memorizzata in un dato vettore appartiene allo stesso tipo di dati. Gli elementi vengono identificati e recuperati attraverso una sequenza di *indici* numerici. L'istruzione DIM che dichiara le caratteristiche di memorizzazione di un vettore è il primo argomento di questo capitolo. Altre parti di questo capitolo descrivono argomenti attinenti, comprese le funzioni LBOUND e UBOUND e le istruzioni ERASE, OPTION BASE e REDIM. Un *record* rappresenta un gruppo di voci di dati attinenti che però appartengono a diversi tipi. Un'istruzione TYPE dichiara il nome e la struttura dei campi di un tipo record definito dall'utente; poi un'istruzione DIM crea una variabile che appartiene a questo tipo. (Si possono utilizzare anche COMMON, SHARED o STATIC per creare una variabile che appartenga ad un tipo definito dall'utente.) Un vettore di record è una combinazione di queste due strutture di dati. Questo tipo di vettore è ideale per memorizzare i record di un database durante l'esecuzione di un programma. Si vedranno esempi dell'uso di questa struttura dati avanzata in molte voci di questo capitolo e nel programma *Compleanno*, presentato nel Capitolo 28.

DIM

L'istruzione DIM può essere utilizzata per dichiarare molte strutture dati e variabili da usare in un programma, ivi compresi i vettori, le variabili record definite dall'utente nonché le variabili scalari appartenenti ai tipi elementari del QuickBASIC.

Dichiarazione di vettori:

```
DIM SHARED vett(dim) AS tip, ...    ' La clausola AS e SHARED sono
                                    ' facoltative.
                                    ' Sono ammesse molte dichiarazioni.
```

Dichiarazione di variabili record:

```
DIM SHARED varRec AS tipRec, ...    ' SHARED è facoltativo.
                                    ' Sono ammesse molte
                                    ' dichiarazioni.
```

Dichiarazione di altre variabili:

```
DIM SHARED var AS tip, ...          ' SHARED è facoltativo.
                                    ' Sono ammesse molte
                                    ' dichiarazioni.
```

Una sola istruzione DIM può dichiarare una lista di strutture multiple e di variabili, oppure il programma può contenere istruzioni DIM multiple per dichiarare diversi tipi di vettori, record e variabili. I capitoli seguenti descrivono dettagliatamente i tre distinti utilizzi dell'istruzione DIM in QuickBASIC.

DIM per dichiarare i vettori

Come dichiarazione di un vettore, l'istruzione DIM presenta le seguenti caratteristiche:

- Il nome del vettore ed il tipo di valori che conterrà.
- Le dimensioni, la lunghezza di ogni dimensione e l'intervallo di valori interi assunti dagli indici che individuano i singoli elementi del vettore.

- Lo stato del vettore in memoria, cioè se la memoria per il vettore è allocata staticamente (durante la compilazione) o dinamicamente (durante l'esecuzione del programma).
- La visibilità dell'array, cioè se l'array sarà visibile in tutto il modulo corrente (SHARED), oppure solo nella procedura che lo definisce.

Nomi e tipi dei vettori

Ogni elemento memorizzato in un dato vettore appartiene allo stesso tipo di dati. Come accade per le variabili semplici, si può esprimere il tipo di un vettore includendo un appropriato carattere di tipo ($, %, &, ! o #) come carattere finale del nome del vettore. In alternativa, si può esprimere il tipo di un vettore con una clausola AS, utilizzando una delle seguenti parole chiave per identificare il tipo: STRING, INTEGER, LONG, SINGLE o DOUBLE. Creare un vettore di record è un processo a due fasi: prima si scrive un'istruzione TYPE per dichiarare un tipo di record definito dall'utente; poi si specifica il nome di questo tipo di record nella clausola AS dell'istruzione DIM che definisce il vettore.

Dimensioni del vettore e intervalli degli indici

Le dimensioni di un vettore determinano il numero di elementi che lo stesso può avere ed il modo in cui gli elementi sono organizzati. Si può esprimere la lunghezza di una data dimensione come un valore intero literal, una variabile o un'espressione. Per default, l'indice di partenza di una data dimensione è 0. Ad esempio, se si esprime una lunghezza di 20, QuickBASIC assegna uno spazio per 21 elementi con un intervallo degli indici da 0 a 20. (Il comando OPTION BASE può cambiare l'indice di partenza di default a 1. Per ulteriori dettagli vedere la voce OPTION BASE.) QuickBASIC consente anche di esprimere uno specifico intervallo di valori per l'indice di una data dimensione, nella forma:

```
iniz TO fine
```

dove *iniz* e *fine* sono valori interi literal, variabili o espressioni e *iniz* è inferiore o uguale a *fine*. I valori di *iniz* e *fine* possono essere negativi o positivi.

Il valore consentito più piccolo è -32.768 ed il più grande è 32.767; comunque la differenza massima possibile tra *iniz* e *fine* è 32.766. Un vettore unidimensionale, analogo ad una singola fila o colonna di valori di dati, viene dichiarato secondo una delle seguenti forme:

```
DIM nomeVett(lungh)
DIM nomeVett(iniz TO fine)
```

Allo stesso modo, una dichiarazione per un vettore bidimensionale, che rappresenta una tabella di valori con più file e colonne, potrebbe presentarsi in una delle seguenti forme:

```
DIM nomeVett(lungh1, lungh2)
DIM nomeVett(iniz1 TO fine1, iniz2 TO fine2)
```

Per esteso, un vettore tridimensionale rappresenta l'unione di due tabelle bidimensionali. In QuickBASIC un vettore può avere fino a sessanta dimensioni. Ogni dimensione separata viene espressa in una delle due rappresentazioni utilizzabili, cioè come un singolo valore che rappresenta una lunghezza o come un intervallo di valori per l'indice. Una volta che un vettore è definito, si individuano singoli elementi del vettore attraverso l'uso di *indici* numerici racchiusi tra parentesi dopo il nome del vettore. Il numero di indici è sempre uguale al numero di dimensioni definite per il vettore stesso. Ogni indice può presentarsi come un valore numerico literal, una variabile o un'espressione. Ad esempio, ognuno dei seguenti nomi rappresenta un elemento di un vettore bidimensionale chiamato *vett2:*

```
vett2(5, 7)
vett2(d1, d2)
vett2(x + 5, 2 * y - 7)
```

Il nome di un elemento del vettore può comparire, come quello di una variabile scalare, in ogni espressione o istruzione BASIC, compresa un'istruzione di assegnamento (si troveranno degli esempi più avanti nel capitolo).

Vettori statici e dinamici

La memoria necessaria per un vettore *statico* è allocata nel momento in cui si compila il programma, mentre un vettore *dinamico* è allocato durante l'esecuzione del programma. Si usa un vettore dinamico nelle seguenti situazioni:

- Quando la lunghezza del vettore si basa su informazioni che diventano disponibili solo durante l'esecuzione del programma.
- Si prevede di dover cambiare la lunghezza di un vettore una o più volte durante un'esecuzione (usando l'istruzione REDIM).
- Si ha bisogno di liberare lo spazio di memoria occupato da un vettore (usando l'istruzione ERASE) dopo che il programma ha finito di utilizzare il vettore.

Si può specificare la staticità o la dinamicità di un vettore nell'istruzione DIM. In generale, quando si è dichiarato che un vettore ha dimensioni numeriche literal, si intende che è statico, mentre un vettore le cui dimensioni sono indicate con nomi di variabile è dinamico. In alternativa, si possono usare le metaistruzioni di QuickBASIC $STATIC e $DYNAMIC per specificare lo stato di default dei vettori. (Vedere il Capitolo 5 "Commenti e metacomandi", nella Parte II, per la sintassi di questi metacomandi.) Infine, si può creare un vettore dinamico dichiarando il suo nome in un'istruzione COMMON e poi stabilendo la sua lunghezza e le sue dimensioni più tardi nel programma. (Vedere la voce REDIM per dettagli ed esempi.) Un vettore *implicitamente dimensionato* è una struttura che non viene effettivamente dichiarata in un'istruzione DIM. In questo caso, ogni dimensione ha indici con intervallo da 0 a 1 ed il vettore è statico per default. L'uso di vettori implicitamente dimensionati è ammesso in QuickBASIC, ma generalmente non è una buona regola di programmazione.

Il campo di visibilità di un vettore

La parola chiave SHARED nell'istruzione DIM crea un vettore che è utilizzabile globalmente in un dato modulo. Questo significa che ogni sottoprogramma o funzione del modulo può accedere al vettore, senza dover

ricevere il vettore come un argomento. (Per informazioni sul passaggio di vettori come argomenti vedere le voci CALL, FUNCTION e SUB nel Capitolo 6.) La parola chiave SHARED è impiegata anche nelle istruzioni DIM che dichiarano altri tipi di variabili, compresi i record.

DIM per dichiarare i record

Si può usare l'istruzione DIM per dichiarare una variabile di tipo record dopo aver definito una struttura di tipo record in un'istruzione TYPE. L'istruzione TYPE definisce il nome di un tipo di record definito dall'utente e specifica i nomi ed i tipi degli elementi del campo che il tipo di record conterrà. (Vedere la voce TYPE per ulteriori dettagli.) Ci si può riferire al nome di questo tipo nella clausola AS dell'istruzione DIM, per creare una variabile di tipo record:

```
DIM varRec AS tipRec
```

Altre istruzioni QuickBASIC che si possono usare per creare una variabile record comprendono REDIM, COMMON, SHARED e STATIC.

DIM per dichiarare altri tipi di variabili

I programmatori BASIC in genere definiscono il tipo di una variabile semplice includendo un carattere tipo ($, %, &, ! o #) alla fine del nome della variabile. L'istruzione DIM fornisce un'alternativa a questo approccio. Si può usare DIM con una clausola AS per dichiarare il tipo di ogni variabile semplice. La clausola AS specifica uno dei seguenti tipi: STRING, INTEGER, LONG, SINGLE o DOUBLE. In particolare, una variabile stringa a lunghezza fissa viene definita come segue:

```
DIM var AS STRING * lungh
```

dove *lungh* è il numero di caratteri che la variabile contiene. Una dichiarazione come questa è richiesta per definire una variabile stringa a lunghezza fissa. Per default, un nome che semplicemente finisce con un carattere $ rappresenta una stringa a lunghezza variabile.

Alcuni esempi dell'istruzione DIM

I seguenti esempi dell'istruzione DIM sono presi dal programma *Compleanno*, presentato nel Capitolo 28 e dal programma *Agenda*, presentato nel Capitolo 30. Entrambi i programmi usano un gruppo di tre valori unidimensionali per memorizzare stringhe ed intere informazioni che descrivono i mesi dell'anno e i giorni della settimana:

```
DIM SHARED nomiMese$(12), giorMese%(12), giorSett$(7)
```

Il tipo di valori di dati che può essere memorizzato in ognuno di questi vettori è specificato dal carattere di tipo alla fine di ogni nome: $ per i vettori stringa e % per il vettore intero. Come indicato dalla parola chiave SHARED, questi tre vettori sono globali. Questa caratteristica è vantaggiosa in programmi che necessitano l'uso di alcuni vettori centrali in più di una procedura. Si noti inoltre che questi tre vettori sono statici: la lunghezza di ogni vettore è espressa come un intero literal. I due programmi non hanno bisogno di ridimensionare questi vettori durante un'esecuzione. È interessante vedere come i programmi memorizzano i dati in questi tre vettori. In ogni caso, un ciclo FOR è utilizzato per leggere i dati da una serie di istruzioni DATA. Ad esempio, il seguente ciclo legge le abbreviazioni di giorni nel vettore *giorSett$*:

```
FOR i% = 1 TO 7
   READ giorSett$(i%)
NEXT i%
```

Ecco la riga DATA che fornisce i sette valori stringa memorizzati nel vettore:

```
DATA Dom, Lun, Mar, Mer, Gio, Ven, Sab
```

Questo semplice passaggio illustra l'importante rapporto che spesso esiste tra i cicli FOR ed i vettori. Un programma può usare la variabile di conteggio di un ciclo FOR (*i%* in questo esempio) come la variabile dell'indice in un vettore. Grazie a questa organizzazione, ogni iterazione del ciclo ha accesso ad un nuovo elemento del vettore. Con solo alcune righe di codice, un programma può elaborare efficientemente un'intera lista di valori di dati. Per esteso, un paio di cicli nidificati possono essere usati per memorizzare

o per leggere i dati in un vettore bidimensionale. Il programma *Compleanno* crea un altro importante vettore, chiamato *staff*, per memorizzare un intero archivio di record dei dipendenti. Dapprima il programma definisce un tipo di record chiamato *dip:*

```
TYPE dip
   Cogn AS STRING * 9      ' Cognome del dipendente.
   Nome  AS STRING * 9     ' Nome del dipendente.
   qualif AS STRING * 18   ' Qualifica del dipendente.
   anniAnz AS SINGLE       ' Anni di anzianità.
   etàAtt AS INTEGER       ' Età del dipendente.
   meseNasc AS INTEGER     ' Mese di nascita.
   compl AS INTEGER        ' Giorno di nascita.
   giorAnno AS INTEGER     ' Intero che rappresenta
                           ' il giorno di compleanno nel calendario.
END TYPE
```

Poi, proprio prima dell'istruzione DIM per il vettore *staff*, un'istruzione READ inizializza il valore della variabile *dimStaff%*:

```
READ dimStaff%
```

La variabile intera rappresenta le dimensioni dell'archivio impiegati (cioè il numero di record). La lunghezza del vettore *staff* è determinata dal valore di questa variabile:

```
DIM SHARED staff(dimStaff%) AS dip
```

Il vettore *staff* è la visibilità globale. Comunque, diversamente dagli altri vettori usati nel programma *Compleanno*, *staff* è un vettore dinamico la cui lunghezza è determinata durante l'esecuzione del programma. Si noti anche che ogni elemento di *staff* appartiene al tipo di dati definito dall'utente chiamato *dip*. Questo implica che ogni elemento di *staff* è un record che contiene tutti gli otto campi di dati definiti nel tipo *dip*. Come si può vedere, un vettore di record come *staff* è una struttura comoda per memorizzare molti valori di dati diversi. È importante capire la notazione per identificare i singoli campi in un dato elemento del record di questo vettore. Il seguente ciclo FOR, che legge i valori di dati nel vettore *staff*, dà degli esempi di questa notazione:

```
   FOR i% = 1 TO dimStaff%
      READ staff(i%).cogn
```

```
      READ staff(i%).nome
      READ staff(i%).qualif

' ...

   NEXT i%
```

Il vettore *staff* è indicizzato dalla variabile *i%* in questo ciclo FOR. Poi il nome del vettore indicizzato è seguito da un punto e dal nome di un dato campo, come definito nella struttura di tipo *dip*. (Vedere la voce TYPE per ulteriori informazioni.) Il programma *Agenda* contiene molte altre illustrazioni interessanti di vettori. Il programma visualizza sullo schermo stringhe di informazioni, che indicano gli appuntamenti per ogni ora di una giornata d'affari. Per memorizzare queste informazioni, il programma crea due vettori di stringhe a lunghezza fissa:

```
DIM SHARED matt(7 TO 12) AS STRING * lunghElem
DIM SHARED pomer(13 TO 18) AS STRING * lunghElem
```

Per convenienza e chiarezza, gli intervalli dell'indice in questi due vettori sono designati per rappresentare le ore della mattina (dalle 7 alle 12) e del pomeriggio (dalle 13 alle 18). Le clausole AS nelle istruzioni DIM definiscono queste strutture come vettori di stringhe a lunghezza fissa. (Senza le clausole AS, queste strutture sarebbero definite per default come vettori di stringa a lunghezza variabile.) Il programma *Agenda* crea un tipo di record chiamato *recGior* per gestire i record che vengono letti e scritti su un file di dati ad accesso random. La seguente dichiarazione è un esempio di un'istruzione DIM che crea una variabile di un solo record. La variabile, chiamata *giorno*, appartiene al tipo *recGior*:

```
DIM giorno AS recGior
```

Per ulteriori informazioni sul programma *Agenda* e le sue strutture dati, consultare il Capitolo 30 di questo libro.

Compatibilità

Nel Turbo Basic, la sintassi dell'istruzione DIM comprende le parole chiave facoltative DYNAMIC e STATIC per dichiarare il formato della memoria

di un vettore. Comunque, la parola chiave SHARED non può far parte dell'istruzione DIM in Turbo Basic. Per rendere un vettore visibile ad una procedura, bisogna inserire il nome del vettore in un'istruzione SHARED. Come QuickBASIC, Turbo Basic consente di definire un intervallo intero per gli indici di un vettore. In ogni caso, in Turbo Basic, per indicare l'intervallo, la notazione utilizza due punti, invece della solita parola chiave TO:

```
DIM nomeVett(iniz:fine, iniz:fine, ...)
```

Turbo Basic non supporta strutture record definite dall'utente, stringhe a lunghezza fissa o l'uso della clausola AS per dichiarare i tipi di variabile. I vettori in BASICA sono sempre dinamici; cioè lo spazio di memoria per il vettore è allocato durante l'esecuzione di un programma. BASICA supporta l'istruzione OPTION BASE per cambiare il punto d'inizio degli indici del vettore a 1 invece del valore di default 0. In ogni caso, in BASICA non c'è modo per specificare l'intervallo di valori che può assumere l'indice dei vettori.

ERASE

Il comando ERASE cancella un vettore dinamico dalla memoria o riinizializza tutti i valori di un vettore statico.

```
ERASE vett, ...        'Possono essere elencati più nomi di vettori.
```

Il risultato di ERASE dipende dal fatto che un dato vettore sia dinamico o statico. (Vedere la voce DIM per una completa descrizione delle dichiarazioni di vettori dinamici e statici.) Un vettore dinamico viene cancellato dalla memoria, cioè lo spazio della memoria originariamente allocato per il vettore viene liberato. Un successivo comando DIM può dichiarare nuove lunghezze per le dimensioni del vettore. (Comunque, un nuovo comando DIM non può cambiare il *numero* di dimensioni nel vettore, anche dopo un comando ERASE.) Si noti che l'azione del comando REDIM è equivalente ad ERASE seguito da un'istruzione DIM. In contrasto, l'effetto del comando ERASE su un vettore statico è semplicemente reinizializzare il vettore. Tutti gli elementi di un vettore numerico sono reinizializzati a zero oppure tutti gli

elementi di un vettore stringa sono inizializzati come stringa nulla. La definizione del vettore è fissata ed il vettore rimane in memoria. Non si può ridimensionare un vettore statico.

Un esempio di ERASE

Il seguente breve programma di prova illustra il risultato di ERASE su un vettore statico. Il programma inizia dichiarando un vettore intero unidimensionale chiamato *test%*, assegnando valori iniziali ai suoi stessi elementi e visualizzando i valori sullo schermo. Poi il programma emette un comando ERASE e rivisualizza i valori del vettore:

```
DIM test%(10)

CLS
FOR i% = 1 TO 10
   test%(i%) = i%
   PRINT test%(i%);
NEXT i%
PRINT

ERASE test%

FOR i% = 1 TO 10
   PRINT test%(i%);
NEXT i%
```

Ecco il risultato che si ottiene, così come viene visualizzato sullo schermo:

```
1 2 3 4 5 6 7 8 9 10
0 0 0 0 0 0 0 0 0 0
```

Si noti che il secondo ciclo FOR del programma visualizza un riga di zeri, i valori reinizializzati del vettore dopo il comando ERASE. È possibile sperimentare ulteriormente questo programma inserendo il seguente metacomando all'inizio del listato:

```
' $DYNAMIC
```

Questo comando specifica che il vettore *test%* sarà dichiarato come una struttura dinamica. (Consultare la voce REM per informazioni circa la sintassi dei metacomandi.) Se non si apportano altri cambiamenti nel

programma, il primo ciclo visualizzerà sullo schermo i valori iniziali del vettore (da 1 a 10), ma il secondo ciclo causerà un errore run-time: "Subscript out of range". Questo accade perché il comando ERASE ha cancellato il vettore dinamico dalla memoria.

Compatibilità

Il Turbo Basic supporta il comando ERASE ma non quello REDIM. Per questo motivo si può usare ERASE seguito da DIM per ridimensionare un vettore dinamico. L'effetto di ERASE su un vettore statico è lo stesso che in QuickBASIC. Tutti i vettori in BASICA sono dinamicamente allocati. ERASE cancella sempre un vettore dalla memoria.

LBOUND

La funzione LBOUND prende il nome di un vettore esistente come suo primo argomento e ritorna un intero che rappresenta l'indice più basso di una specifica dimensione nel vettore.

```
LBOUND(vett)        ' Per un vettore unidimensionale.
```

o

```
LBOUND(vett, dim)   ' Per un vettore multidimensionale.
```

Il nome che appare come il primo argomento della funzione LBOUND rappresenta un vettore che è già stato esplicitamente dichiarato in un'istruzione DIM o implicitamente dichiarato attraverso il suo utilizzo in alcune altre istruzioni. Il secondo argomento di LBOUND non è richiesto se il vettore ha solo una dimensione; in questo caso, LBOUND fornisce semplicemente l'indice più basso del vettore. Ad esempio, dato un vettore unidimensionale chiamato *vett1*, la seguente espressione dà l'indice più basso:

```
LBOUND(vett1)
```

Per vettori multidimensionali, comunque, il secondo argomento deve apparire come un intero positivo che specifica il numero della dimensione interessata. Ad esempio, dato un vettore tridimensionale chiamato *vett3*, la seguente chiamata a LBOUND ritorna l'indice più basso della seconda dimensione:

```
LBOUND(vett3, 2)
```

Le funzioni LBOUND e UBOUND sono utili nelle procedure che devono scoprire l'intervallo dell'indice di un vettore che è passato come un argomento o di un vettore che ha un campo di validità globale.

Un esempio della funzione LBOUND

Il programma *Agenda*, presentato nel Capitolo 30, dichiara un vettore stringa globale chiamato *matt* per memorizzare gli appuntamenti del mattino per una certa data del calendario. L'intervallo dell'indice di questo vettore corrisponde alle ore del mattino dalle 7 a mezzogiorno:

```
DIM SHARED matt(7 TO 12) AS STRING * lunghElem
```

Più tardi nel programma, un sottoprogramma chiamato *VisualGior* ha il compito di visualizzare sullo schermo questi appuntamenti del mattino. Un ciclo FOR *VisualGior* usa le funzioni LBOUND e UBOUND per determinare l'esatto numero di iterazioni per la variabile conteggio *ora%*:

```
FOR ora% = LBOUND(matt) TO UBOUND(matt) - 1
   PRINT USING "##:00 a.m. > " ora%;
   PRINT matt(ora%)
NEXT ora%
```

Insieme le funzioni LBOUND e UBOUND garantiscono che il ciclo produrrà l'intervallo corretto di indici per l'accesso agli elementi del vettore.

Compatibilità

LBOUND e UBOUND sono utilizzabili anche in Turbo Basic, usando lo stesso formato visto per QuickBASIC. Si noti, comunque, che la sintassi per

definire un intervallo di indici per un vettore è leggermente differente in Turbo Basic. (Vedere la voce DIM per ulteriori dettagli.) LBOUND e UBOUND non esistono in BASICA, né BASICA permette di specificare un intervallo di indici nell'istruzione DIM. (BASICA comunque supporta il comando OPTION BASE per cambiare l'indice inferiore da 0 a 1.)

OPTION BASE

L'istruzione OPTION BASE stabilisce l'indice di partenza di default dei vettori come 0 o 1.

```
OPTION BASE 0
```

o

```
OPTION BASE 1
```

Il valore OPTION BASE deve comparire come un valore intero literal, o 0 o 1. (L'editor "intelligente" di QuickBASIC non accetta una variabile o un'espressione come una sintassi valida per questo comando.) L'istruzione DIM dà due diversi modi per esprimere la lunghezza di una dimensione o l'intervallo di valori del corrispondente indice, *lungh*, o un intervallo di interi, *iniz TO fine*. (Vedere la voce DIM per ulteriori dettagli.) Se si esprime una lunghezza di dimensione come un singolo intero, l'indice di default più basso è 0. Ad esempio, la seguente istruzione DIM dichiara un vettore intero uni-dimensionale chiamato *test%*, il cui intervallo di indici va da 0 a 10:

```
DIM test%(10)
```

In ogni modo, si può cambiare questo default includendo un'istruzione OPTION BASE all'inizio del proprio programma; ad esempio:

```
OPTION BASE 1
DIM test%(10)
```

Ora il vettore di *test%* ha indici con intervalli che vanno da 1 a 10. Se si inserisce in un programma, OPTION BASE appare nel programma principale (o codice "modulare"), prima che venga dichiarato qualsiasi vettore.

In molte applicazioni però è preferibile usare la notazione *iniz To fine* nell'istruzione DIM per esprimere l'intervallo dei valori dell'indice di un dato vettore. Ad esempio, la seguente dichiarazione (presa dal programma *Agenda* presentato nel Capitolo 30) crea un vettore chiamato *pomer*, il cui intervallo di indice va da 1 a 6:

```
DIM SHARED pomer(1 TO 6) AS STRING è lunghElem
```

La singola istruzione è più chiara e più economica di queste due:

```
OPTION BASE 1
DIM SHARED pomer(6) AS STRING * lunghElem
```

La presenza di un'istruzione OPTION BASE in un programma non impedisce di usare la notazione *iniz TO fine* in un'istruzione DIM; questa notazione ha sempre la precedenza sull'indice iniziale stabilito da OPTION BASE.

Compatibilità

Turbo Basic supporta un'istruzione OPTION BASE che consente di stabilire qualsiasi valore intero come indice iniziale, non solo 0 o 1. Turbo Basic supporta anche una notazione per specificare l'intervallo di valori dell'indice nell'istruzione DIM. (Consultare la voce DIM per ulteriori informazioni.) L'istruzione OPTION BASE in BASICA è la stessa di QuickBASIC.

REDIM

L'istruzione REDIM rialloca lo spazio per uno o più vettori dinamici, cambiando potenzialmente le lunghezze delle dimensioni. REDIM inizializza tutti gli elementi di ogni vettore che dichiara; tutti i dati memorizzati precedentemente nel vettore vengono persi.

```
REDIM SHARED vett(dimens) AS TIPO, ...  ' Le clausole AS e SHARED
                                        ' sono facoltative.
                                        ' Sono ammesse più
                                        ' dichiarazioni.
```

Una sola istruzione REDIM può indicare una lista di più vettori oppure il programma può contenere più istruzioni REDIM per riallocare singoli vettori. Come l'istruzione DIM, REDIM specifica le caratteristiche del vettore o dei vettori che rialloca. Specificatamente, REDIM fornisce il nome di ogni vettore ed indica il tipo di valori che il vettore conterrà. Il tipo di dati può essere espresso tramite il carattere finale del nome del vettore ($, %, &, ! o #), oppure in una possibile clausola AS che dà il nome del tipo di dati (STRING, INTEGER, LONG, SINGLE, DOUBLE o un tipo di record definito dall'utente). REDIM specifica anche la lunghezza o l'intervallo degli indici per ogni dimensione. Come nell'istruzione DIM, si può esprimere la lunghezza come un valore intero literal, una variabile o un'espressione; altrimenti si può dare l'effettivo intervallo di indici per una data dimensione, nella forma *iniz TO fine*. (Vedere la voce DIM per ulteriori dettagli.) REDIM non può cambiare il numero di dimensioni in un vettore esistente, ma solo le lunghezze delle esistenti dimensioni. Ad esempio, se si sta ridimensionando un vettore esistente bidimensionale, si può cambiare la lunghezza di ogni dimensione, ma non si può ridefinire la struttura come un vettore tridimensionale. La parola facoltativa SHARED nell'istruzione REDIM crea un vettore che è utilizzabile globalmente in un dato modulo. Ciò significa che ogni sottoprogramma o funzione nel modulo può accedere al vettore, senza dover ricevere il vettore come un argomento. Si tenga presente l'effetto di REDIM su un vettore esistente: i valori correnti dei dati del vettore vengono persi. QuickBASIC inizializza gli elementi del vettore al valore nullo in un vettore stringa o a 0 in un vettore numerico. Se si ha bisogno di riutilizzare i valori memorizzati in un vettore esistente, bisogna copiare i valori su un'altra struttura prima di eseguire il comando REDIM.

Alcuni esempi dell'istruzione REDIM

Il programma *Agenda*, presentato nel Capitolo 30, mantiene un archivio per memorizzare una tabella giornaliera personale. Il programma usa un vettore chiamato *indData* come strumento per localizzare specifici record nel file dati ad accesso random che memorizza il calendario. In effetti, il vettore *indData* serve come indice nel file di dati, fornendo le locazioni di specifici record memorizzati nel file. (Per ulteriori dettagli consultare il Capitolo 30 in cui vi è una descrizione completa del programma.) La struttura di *indData*

è un vettore dinamico, globale, uni-dimensionale di record. La prima dichiarazione del vettore è la seguente istruzione COMMON SHARED:

```
COMMON SHARED contElem%, indData() AS elemInd
```

Questa istruzione dà al nome del vettore una visibilità globale, ma non alloca effettivamente la memoria per il vettore. (Vedere la voce COMMON SHARED per maggiori particolari.) Per determinare la lunghezza appropriata del vettore, il programma calcola il numero di record corrente del database; questo valore viene memorizzato nella variabile intera *contElem%*. Poi, la seguente istruzione REDIM alloca lo spazio per il vettore *indData*:

```
REDIM indData(contElem%) AS elemInd
```

I record vengono poi letti in *indData* da un file sequenziale memorizzato su disco. Quando l'immissione interattiva dell'utente assegna un record della nuova tabella giornaliera, la lunghezza del vettore *indData* deve essere aumentata di un record. Il programma aumenta il valore di *contElem%* di 1, e poi copia attentamente l'intera serie di record indice in un vettore temporaneo chiamato *indProvv*. Poi il vettore *indData* viene ridimensionato:

```
REDIM indData(contElem%) AS elemInd
```

Dopo questa nuova allocazione di spazio di memoria per il vettore, il programma copia l'indice precedente da *indProvv* in *indData*. Infine, il programma memorizza un nuovo record di indice nel nuovo elemento del vettore e poi ordina il vettore così che possa continuare a servire come una valida struttura di indice.

Compatibilità

Né BASICA né Turbo Basic supportano l'istruzione REDIM. Comunque entrambe consentono il ridimensionamento dei vettori, attraverso queste due fasi:

1. Usare il comando ERASE per disallocare dalla memoria un vettore dinamico esistente.

2. Usare il comando DIM per riallocare spazio per lo stesso vettore, ma con la lunghezza desiderata.

TYPE

L'istruzione TYPE dichiara il tipo definito dall'utente per l'utilizzo in un programma QuickBASIC. Un'istruzione TYPE può comparire solo nel programma principale (o "modulare"). TYPE definisce solo un tipo di struttura, non una variabile. In genere un'istruzione DIM segue l'istruzione TYPE per creare una variabile record che appartiene al tipo definito dall'utente.

```
TYPE nomeTip
   nomeCamp1 AS tip
   nomeCamp2 AS tip

'     ...              Un numero arbitrario di campi possono essere
'                      inclusi nella definizione del tipo.
END TYPE
```

Il nome di un tipo definito dall'utente inizia con una lettera ed è formato al massimo da 40 lettere e cifre. Tra le righe TYPE e END TYPE si possono definire qualsiasi numero di elementi per la struttura. Una struttura definita dall'utente viene chiamata anche *tipo di record* e gli elementi della struttura sono chiamati *campi*. L'importanza di una struttura record è che può contenere campi che appartengono ad una varietà di diversi tipi di dati. La clausola AS specifica il tipo di dati di ogni campo nella definizione TYPE. Ogni campo può appartenere ad uno dei seguenti tipi di QuickBASIC: il tipo stringa a lunghezza fissa (STRING * lunghezza), INTEGER, LONG, SINGLE o DOUBLE. Inoltre, un campo può appartenere ad un altro tipo di record definito precedentemente. Un campo non può essere né un vettore né una stringa a lunghezza variabile. Una volta definita una struttura TYPE, si può dichiarare un numero arbitrario di variabili e/o vettori che appartengono al tipo. Ognuna delle seguenti istruzioni può essere usata per creare una variabile di record o un vettore di record: DIM, REDIM, COMMON, SHARED o STATIC. Si possono definire globalmente variabili record, per utilizzarle in un intero modulo di programma o localmente per usarle all'interno di un solo sottoprogramma o funzione. Data una variabile di tipo record, la seguente notazione identifica nella struttura un singolo campo:

```
nomeVar.nomeCamp
```

Allo stesso modo, questa notazione identifica un solo campo nell'elemento *i%* di un vettore di record:

```
nomeVett(i%).nomeCamp
```

Consultare la voce DIM per ulteriori dettagli ed esempi.

Un esempio dell'istruzione TYPE

Il programma *Compleanno*, presentato nel Capitolo 28, definisce una struttura di record, chiamata *dip*, per memorizzare i vari dati riguardanti gli impiegati. Ecco la definizione TYPE per questa struttura:

```
TYPE dip
   Cogn AS STRING * 9      ' Cognome del dipendente.
   Nome  AS STRING * 9     ' Nome del dipendente.
   qualif AS STRING * 18   ' Qualifica del dipendente.
   anniAnz AS SINGLE       ' Anni di anzianità.
   etàAtt AS INTEGER       ' Età del dipendente.
   meseNasc AS INTEGER     ' Mese di nascita.
   compl AS INTEGER        ' Giorno di nascita.
   giorAnno AS INTEGER     ' Intero che rappresenta
                           ' il giorno di compleanno nel calendario.
END TYPE
```

Si noti che la struttura contiene campi che appartengono a vari tipi di dati: tre stringhe a lunghezza fissa, un numero in virgola mobile in precisione semplice e quattro interi. In seguito nel programma *Compleanno*, la seguente istruzione DIM definirà un vettore di record chiamato *staff*, che appartiene al tipo *dip:*

```
DIM SHARED staff(dimStaff%) AS dip
```

Ogni elemento del vettore *staff* è perciò un record di tipo *dip*, che contiene tutti e otto i campi definiti. Questa struttura è ideale per memorizzare un intero database degli impiegati.

Compatibilità

Né Turbo Basic né BASICA supportano tipi di record definiti dall'utente.

UBOUND

La funzione UBOUND prende il nome di un vettore esistente come suo primo argomento e ritorna un intero che rappresenta l'indice più alto di una specifica dimensione del vettore.

```
UBOUND(vett)         ' Per un vettore uni-dimensionale.
```

o

```
UBOUND(vett, dim)    ' Per un vettore multidimensionale.
```

Il nome che compare come primo argomento della funzione UBOUND rappresenta un vettore che è già stato dichiarato esplicitamente in un'istruzione DIM o implicitamente attraverso il suo utilizzo in alcune altre istruzioni. Il secondo argomento di UBOUND non è richiesto se il vettore contiene solo una dimensione; in questo caso, UBOUND fornisce semplicemente l'indice più alto del vettore. Ad esempio, dato un vettore unidimensionale chiamato *vett1*, la seguente espressione dà l'indice più alto:

```
UBOUND(vett1)
```

Per un vettore multidimensionale, comunque, il secondo argomento deve apparire come un intero positivo che specifica il numero della dimensione interessata. Ad esempio, dato un vettore tridimensionale chiamato *vett3*, la chiamata a UBOUND ritorna l'indice più alto della seconda dimensione:

```
UBOUND(vett3, 2)
```

Le funzioni UBOUND e LBOUND sono utili nelle procedure che devono scoprire l'intervallo dell'indice di un vettore che è passato come un argomento o di un vettore con campo di visibilità globale.

Un esempio della funzione UBOUND

Il programma *Agenda*, presentato nel Capitolo 30, dichiara un vettore stringa globale con nome *pomer* per memorizzare gli appuntamenti di una certa data. L'intervallo di valori dell'indice di questo vettore corrisponde alle ore del pomeriggio dalle 13 alle 18:

```
DIM SHARED pomer(13 a 18) AS STRING * lunghElem
```

Un sottoprogramma chiamato *VisualGior* ha il compito di visualizzare su schermo questi appuntamenti del pomeriggio. Un ciclo FOR in *VisualGior* perciò usa le funzioni UBOUND e LBOUND per determinare l'intervallo corretto di iterazioni per la variabile di conteggio *ora%*:

```
FOR ora% = LBOUND(pomer) TO UBOUND(pomer)
   PRINT USING " #:00 p.m. >"; ora%;
   PRINT pomer(ora%)
NEXT ora%
```

Insieme, le funzioni UBOUND e LBOUND garantiscono che il ciclo fornirà l'intervallo corretto di indici per accedere agli elementi del vettore.

Compatibilità

LBOUND e UBOUND sono disponibili anche nel Turbo Basic, usando lo stesso formato del QuickBASIC. Si noti comunque che la sintassi per definire l'intervallo degli indici per un vettore è leggermente diverso nel Turbo Basic. (Vedere la voce DIM per ulteriori dettagli.) LBOUND e UBOUND non esistono in BASICA, né BASICA permette di specificare un intervallo di indici nell'istruzione DIM. (BASICA comunque non supporta il comando OPTION BASE per cambiare l'indice più basso da 0 a 1.)

Capitolo 4

Assegnamenti

L'istruzione di assegnamento, scritta con o senza la parola chiave LET, è un modo di memorizzare un valore in una variabile QuickBASIC. La voce LET di questo capitolo descrive i vari utilizzi dell'istruzione di assegnamento. Sono disponibili anche molte altre tecniche per gestire dati e variabili, che dipendono dal contesto della programmazione. Ad esempio, si può usare l'istruzione READ per leggere dei dati da una sequenza di righe di DATA e memorizzarli in alcune variabili. READ, DATA e l'istruzione collegata RESTORE vengono trattate in questo capitolo. Inoltre, QuickBASIC consente tramite l'istruzione SWAP di *scambiare* i valori memorizzati nelle due variabili che appartengono allo stesso tipo. Questa istruzione è utile in ogni routine di ordinamento che si scrive in QuickBASIC. In questo capitolo SWAP è la voce finale.

DATA

Un'istruzione DATA memorizza una lista di dati all'interno del listato di un programma QuickBASIC. Con una serie di istruzioni READ si possono leggere i valori di questi dati in sequenza ed assegnarli a delle variabili.

```
DATA lisDati
```

La lista di dati in un'istruzione DATA può contenere tutte le combinazioni di valori stringa e numerici. Ognuno è separato dal successivo da una

virgola. Un valore stringa non deve essere tra parentesi, a meno che la stringa stessa inizi o finisca con spazi, o contenga virgole o due punti. Non c'è un limite per il numero di valori compresi nella lista di dati, però l'intera istruzione DATA deve stare su una sola riga nel listato del programma. Un programma QuickBASIC può contenere qualsiasi numero di istruzioni DATA, tutte localizzate nel programma principale o "codice a livello del modulo". (L'editor QuickBASIC sposta automaticamente le istruzioni DATA nel programma principale, se le si immette in un sottoprogramma o in una funzione.) Il programma può utilizzare successivamente una serie di istruzioni READ per leggere dati in alcune variabili. Per default, le liste di valori DATA vengono lette sequenzialmente tramite un puntatore che scandisce i vari valori, dalla prima istruzione DATA all'ultima. Comunque, l'istruzione RESTORE riposiziona il puntatore all'inizio della prima istruzione DATA o in un punto specifico nel listato del programma, così che le successive istruzioni READ possono rileggere tutti o parte dei dati. (Vedere le voci READ e RESTORE per ulteriori dettagli.)

Alcuni esempi di istruzioni DATA

Un'istruzione DATA memorizza dati stringa o numerici che nella struttura completa di un programma servono a scopi generali. DATA e READ sono utili soprattutto per inizializzare dei vettori di dati. Ad esempio, le seguenti istruzioni DATA sono prese dal programma *Compleanno* presentato nel Capitolo 28. Le istruzioni memorizzano alcune informazioni relative al calendario, compresi i nomi abbreviati dei mesi e giorni e il numero di giorni in ogni mese:

```
DATA Gen, 31, Feb, 29, Mar, 31, Apr, 30, Mag, 31, Giu, 30
DATA Lug, 31, Ago, 31, Set, 30, Ott, 31, Nov, 30, Dic, 31
DATA Dom, Lun, Mar, Mer, Gio, Ven, Sab
```

All'interno di un sottoprogramma chiamato *LeggiInfCal*, un paio di cicli FOR possono leggere efficacemente tutte queste informazioni in tre vettori, *nomiMese$, giorMese%, giorSett$*:

```
FOR i% = 1 TO 12
   READ nomiMese$(i%)
   READ giorMese%(i%)
```

```
NEXT i%
FOR i% = 1 TO 7
   READ giorSett$(i%)
NEXT i%
```

I programmatori usano talvolta l'istruzione DATA per memorizzare i record di uno specifico database, ad esempio nomi, indirizzi, numeri di telefono e così via, all'interno del listato di un programma. Si tratta di una pratica molto discutibile; infatti, questo tipo di dati dovrebbe normalmente essere memorizzato su disco in un file di dati a parte. Se si memorizzano i dati nel listato di un programma, bisogna ricompilare il programma ogni volta che si apportano dei cambiamenti agli stessi dati. Non di meno, per le applicazioni di programma che lavorano con database piccoli e relativamente stabili, l'istruzione DATA può essere ancora lo strumento più conveniente per memorizzare le informazioni. Ad esempio, il programma *Compleanno* lavora con un piccolo database degli impiegati che è memorizzato in questa sequenza di istruzioni DATA:

```
'     Nome            Qualifica     Data Assun. Data Nasc.

DATA Billi,   Anna,    Segretaria,   8, 1, 1988, 6, 4, 1966
DATA Rossi,   Paolo,   Impiegato,    7, 3, 1985, 7, 7, 1955
DATA Poli,    Luca,    Ragioniere,   1, 2, 1987, 1, 1, 1959
DATA Aimi,    Carla,   Impiegata,    3, 5, 1983, 5, 9, 1950
DATA Telli,   Rosa,    Dirigente,    5, 2, 1983, 9, 9, 1952
DATA Dadda,   Marco,   Dir. Vend.,   1, 3, 1989, 4, 3, 1946
DATA Frosi,   Mara,    Rappresent., 2, 4, 1988, 5, 8, 1965
DATA Mari,    Diego,   Programmat., 6, 2. 1989, 8, 8, 1966
DATA Vago,    Mario,   Ricercatore, 2, 5, 1988, 7, 9, 1963
DATA Pini,    Clara,   Segretaria,   1, 9, 1987, 2, 2, 1964
DATA Orio,    Tina,    Rappresent., 1, 9, 1987, 3, 5, 1959
DATA Gosi,    Milo,    Programmat., 4, 6, 1986, 1, 3, 1968
DATA Sora,    Fabio,   Dirigente,    8, 9, 1984, 6, 6, 1950
DATA Radi,    Lara,    Impiegata,    7, 4, 1988, 4, 1, 1960
DATA Bosio,   Carlo,   Dir. Tecn.,   9, 1, 1987, 7, 3, 1955
```

Il programma può convenientemente leggere e rileggere questo database nei vettori appropriatamente definiti. Se avvengono dei cambiamenti nel database, ad esempio un impiegato cambia compiti o lascia la ditta, oppure viene assunto un nuovo impiegato, il programmatore deve modificare il database direttamente all'interno del listato del programma e poi ricompilare il programma. Un metodo comodo per gestire i dati è che il programmatore e l'utente del database siano la stessa persona. Si può

inserire un commento alla fine di un'istruzione DATA; comunque bisogna utilizzare i due punti per separare l'ultimo dato dal commento. Ad esempio, si consideri questa istruzione del programma *Compleanno*:

```
DATA 15 : REM Numero attuale dei dipendenti.
```

Senza i due punti, il programma leggerebbe il commento come se fosse un dato.

Compatibilità

L'istruzione DATA è disponibile sia in BASICA sia in Turbo Basic, in forma uguale a quella di QuickBASIC. Ad ogni modo, solo l'editor QuickBASIC sposta automaticamente le istruzioni DATA nel programma principale.

LET (ISTRUZIONE DI ASSEGNAMENTO)

Un'istruzione di assegnamento memorizza un solo valore stringa o numerico in una variabile QuickBASIC. La parola chiave LET è facoltativa.

```
LET var = espr
```

o

```
var = espr
```

Il nome a sinistra rispetto al segno di uguale può essere una variabile scalare, un elemento di un vettore, un record o un solo campo di un record. Il valore dopo l'uguale può apparire come valore literal, come variabile o come un'espressione. QuickBASIC valuta l'espressione a destra dell'uguale ed assegna il valore risultante alla variabile specificata. In generale, il tipo della variabile deve corrispondere al tipo di dati del valore che deve esserle assegnato. (Comunque, QuickBASIC può eseguire alcune conversioni di tipo automaticamente in un'istruzione di assegnamento. Vedere la voce CDBL per ulteriori dettagli.) Il nome di una variabile in QuickBASIC inizia

sempre con una lettera dell'alfabeto e può contenere fino a 40 lettere e cifre (è ammesso anche il punto). I caratteri maiuscoli o minuscoli non sono importanti nei nomi delle variabili, sebbene l'editor di QuickBASIC mantenga automaticamente l'uniformità tra tutti gli usi di una data variabile, in un modulo del programma. Il carattere finale del nome dichiara il tipo della variabile: %, &, !, # o $. In alternativa, le istruzioni DEF*tipo* consentono di dichiarare prima i tipi di variabili che iniziano con lettere specifiche dell'alfabeto. (Vedere la voce DEF*tipo* per ulteriori dettagli.) Infine, si può utilizzare anche l'istruzione DIM con una clausola AS per dichiarare il tipo di ogni variabile. (Vedere la voce DIM.) Una sola istruzione di assegnamento non può assegnare ad un vettore più valori contemporaneamente, ma un solo valore alla volta per elemento del vettore. Invece, QuickBASIC può copiare con successo tutti i valori dei campi di un record in un altro. Nell'istruzione di assegnamento che esegue questo compito, le variabili del record dello stesso tipo definito dall'utente si presentano sia a destra sia a sinistra del segno di uguale.

Alcuni esempi di istruzioni di assegnamento

Tutti questi esempi sono presi dal programma *Compleanno*, presentato nel Capitolo 28. La forma più semplice dell'assegnamento è un'istruzione che memorizza un valore literal in una variabile. Ad esempio, questa istruzione *inizializza* la variabile *temp%* a zero:

```
temp% = 0
```

L'espressione alla destra del segno di uguale può contenere un numero qualsiasi di elementi, compresi valori literal, costanti simboliche, variabili, operazioni (aritmetiche, relazionali e logiche) e chiamate di funzione. Ad esempio, la seguente espressione contiene due variabili, due operazioni aritmetiche, un valore numerico literal ed una chiamata di funzione; il risultato viene assegnato a *CalcEtà%*:

```
CalcEtà% = CINT((annoCorr& - dataNasc&) / 365)
```

A questo proposito, va detto che questo particolare assegnamento si trova alla fine della funzione chiamata *CalcEtà%*; il ruolo specifico dell'istru-

zione è definire il valore di ritorno della funzione. La variabile nominata alla sinistra del segno di uguale può comparire anche nell'espressione sulla destra. Ad esempio, la seguente istruzione aumenta il valore della variabile *temp%* del numero memorizzato nell'elemento del vettore *giorMese%(i%)*:

```
temp% = temp% + giorMese%(i%)
```

L'espressione sulla destra del segno di uguale può includere anche l'operazione di confronto logico rappresentata dal segno di uguale; ad esempio, si consideri quest'istruzione:

```
divPer4% = (anno% MOD 4 = 0)
```

Quest'istruzione assegna il valore logico vero o falso alla variabile *divPer4%*. Se l'espressione *anno% MOD 4* è zero, allora *divPer4%* riceve il valore vero; altrimenti *divPer4%* è falso. (In alcuni programmi è importante sapere come QuickBASIC rappresenta i valori logici vero e falso: falso è rappresentato come valore intero 0 e vero è -1. Viceversa, nel contesto di un'espressione logica, QuickBASIC valuta 0 come un falso logico ed ogni altro intero come vero.) La variabile a sinistra dell'uguale può essere un elemento di un vettore o un campo di una variabile record definita dall'utente. Ad esempio, l'istruzione seguente assegna un valore al campo *etàAtt* del record *staff(i%)*:

```
staff(i%).etàAtt = CalcEtà%(meseNasc%, giorNasc%, annoNasc%)
```

In quest'istruzione, *staff* è un vettore di record. Inoltre, se *rec1* e *rec2* sono variabili record dello stesso tipo, la seguente istruzione copia tutti i valori dei campi di *rec2* nei campi corrispondenti di *rec1*:

```
rec1 = rec2
```

Questa potente istruzione illustra uno dei maggiori vantaggi del tipo record definito dall'utente in QuickBASIC.

Compatibilità

Le istruzioni di assegnamento compaiono ovunque nella stessa forma sia in Turbo Basic sia in BASICA, ma nessuno di questi linguaggi supporta tipi record definiti dall'utente.

READ

Un'istruzione READ legge uno o più dati dalle istruzioni DATA ed assegna i valori ad una o più variabili.

```
READ var
```

Un'istruzione READ contiene una lista di una o più variabili QuickBASIC, ognuna separata dalla successiva, nella lista, da una virgola. L'istruzione legge i valori dei dati in sequenza da una qualsiasi istruzione DATA contenuta nel listato del programma ed assegna ciascun valore letto ad una delle variabili della lista. Le istruzioni READ seguenti continuano a leggere uno dopo l'altro i dati memorizzati nelle istruzioni DATA del programma. Se il numero di variabili in una lista READ supera il numero dei valori di dati rimasti disponibili nelle istruzioni DATA, QuickBASIC visualizza il messaggio di errore run-time "Out of DATA". (Comunque, un programma può usare di nuovo l'istruzione DATA così che le informazioni possano essere rilette. Vedere la voce RESTORE per ulteriori dettagli.) Il tipo delle variabili in un'istruzione READ deve essere quello appropriato per ricevere i valori corrispondenti che l'istruzione legge da una riga DATA. QuickBASIC visualizza un messaggio d'errore run-time se un programma tenta di leggere un valore stringa in una variabile numerica. Le variabili in una lista READ possono includere variabili scalari, elementi di un vettore o di campi di una variabile record. Un'istruzione READ non può leggere nello stesso tempo più valori in un vettore o in un record.

Alcuni esempi dell'istruzione READ

Si può usare una sola istruzione READ per leggere più dati nelle loro rispettive variabili. Ad esempio, la seguente istruzione legge degli interi nelle quattro variabili *a%*, *b%*, *c%* e *d%*:

```
READ a%, b%, c%, d%
```

In alternativa, si può scrivere una sequenza di istruzioni READ, ognuna delle quali legge un dato. Ad esempio, si consideri questo pezzo del programma *Compleanno* (presentato nel Capitolo 28):

```
        READ staff(i%).cogn$
        READ staff(i%).nome$
        READ staff(i%).qualif$
        READ meseIniz(i%)
        READ giorIniz(i%)
        READ annoIniz(i%)

'   ...

        READ  meseNasc%
        READ  giorNasc%
        READ  annoNasc%
```

Questa sequenza legge tre valori stringa e sei interi da una sola istruzione DATA. L'istruzione DATA corrispondente contiene una sequenza appropriatamente tipata di valori di dati; ad esempio, ecco una delle istruzioni DATA che il programma legge:

```
DATA Billi, Anna, Segretaria, 8, 1, 1988, 6, 4, 1966
```

Si noti che i primi tre valori sono letti nei campi di un elemento di un vettore di record chiamato *staff*. Gli altri valori vengono assegnati a variabili scalari intere.

Compatibilità

Le istruzioni READ, DATA e RESTORE sono tutte disponibili in BASICA e in Turbo Basic ed il loro uso è lo stesso di quello in QuickBASIC.

RESTORE

L'istruzione RESTORE specifica la riga DATA dalla quale il successivo comando READ inizierà a leggere i dati.

```
RESTORE           ' Inizia a leggere dalla prima
                  ' istruzione DATA nel listato.
```

o

```
RESTORE numRiga   ' Inizia a leggere dalla prima
                  ' istruzione DATA dopo numRiga.
```

o

```
RESTORE eticRiga  ' Inizia a leggere dalla prima
                  ' istruzione DATA dopo eticRiga.
```

Il compito dell'istruzione RESTORE è di consentire ad un programma di rileggere i valori dei dati memorizzati in alcune o in tutte le istruzioni DATA che compaiono nel listato del programma. In effetti, RESTORE sposta il puntatore all'istruzione DATA che sarà letta successivamente. RESTORE da solo, emesso senza un argomento, posiziona il puntatore all'inizio della prima istruzione DATA nel listato del programma. In alternativa, si può includere nell'istruzione RESTORE il numero o l'etichetta di una riga; in questo caso, RESTORE sposta il puntatore sulla prima istruzione DATA dopo la riga specificata. L'istruzione READ successiva inizia a leggere dalla riga DATA indicata.

Un esempio dell'istruzione RESTORE

Il programma *Compleanno*, presentato e discusso nel Capitolo 28, ha due serie distinte di istruzioni DATA. La prima contiene dati relativi al calendario e la seconda è un database composto dai record degli impiegati. All'inizio dell'esecuzione del programma, entrambe le serie di dati vengono lette in strutture appropriate. In particolare, il database degli impiegati viene memorizzato in un vettore di record chiamato *staff*. Seguendo le istruzioni interattive dell'utente, il programma ordina il vettore *staff* rispetto ad uno dei molti campi chiave disponibili. Il sottoprogramma chiamato *Selez* esegue la selezione. Comunque, se l'utente sceglie la sequenza che rispetta l'ordine definito dalle istruzioni DATA originali, cioè l'ordine secondo le date di assunzione degli impiegati, il programma non si preoccupa di ordinare il vettore *staff*. Legge, invece, semplicemente le istruzioni DATA originali nel vettore:

```
IF tastoOrdin% = YEARSORTD THEN
   RESTORE Database
   LeggiRec
   EXIT SUB
END IF
```

In questo caso, l'istruzione RESTORE posiziona il puntatore DATA sulla prima istruzione DATA localizzata dopo l'etichetta *Database*; questa etichetta è localizzata proprio prima del primo record:

```
Database:

'     Nome         Posiz.            Data Iniz.    Data Nasc.
DATA  Billi, Anna,  Segretaria,       8, 1, 1988,   6, 4, 1966
' ...
```

Una volta che il puntatore DATA è stato riposizionato, una chiamata al sottoprogramma *LeggiRec* rilegge il database degli impiegati nel vettore *staff*. Poi un'istruzione EXIT SUB salta il resto del sottoprogramma *Selez*. Il programma *Compleanno* presenta un menù che consente all'utente di vedere il database degli impiegati in tutti i diversi ordini possibili. Grazie all'istruzione RESTORE, il programma può semplicemente rileggere il database dalle istruzioni DATA ogni volta che l'utente seleziona l'ordine originale.

Compatibilità

Sia BASICA sia Turbo Basic supportano l'istruzione RESTORE.

SWAP

L'istruzione SWAP scambia i valori di due variabili che appartengono allo stesso tipo di dati:

```
SWAP var1, var2
```

Le due variabili nell'istruzione SWAP sono separate da una virgola. Devono appartenere entrambe allo stesso tipo di dati: stringa, intero, intero lungo, in

precisione semplice, in doppia precisione, tipo record definito dall'utente. L'istruzione SWAP non può eseguire nessuna conversione di tipo, neanche da un tipo numerico all'altro.

Un esempio di SWAP

SWAP è frequentemente usato nelle procedure di ordinamento, per scambiare due elementi che non sono nell'ordine voluto. Ad esempio, il programma *Compleanno* (discusso nel Capitolo 28) contiene una procedura di ordinamento chiamata *Selez*. All'interno di un paio di cicli nidificati, questa procedura confronta coppie di elementi del vettore di record chiamato *staff*. Se si scopre che una coppia data non rispetta l'ordinamento, alla variabile *swp%* viene assegnato il valore logico vero. Nell'istruzione seguente, SWAP viene eseguito se *swp%* è vero:

```
IF swp% THEN SWAP staff(i%), staff(j%)
```

Si noti che questa istruzione scambia tutti i campi delle variabili record *staff(i%)* e *staff(j%)*.

Compatibilità

Sia BASICA sia Turbo Basic supportano il comando SWAP.

Parte II

Struttura dei programmi e controllo del flusso esecutivo

Nel passato, programmare in BASIC spesso voleva dire scrivere usando uno stile confuso, caratterizzato da un uso smodato dell'istruzione GOTO. I programmi che ne derivavano, nonostante dessero risultati utili o anche meravigliosi, erano spesso illeggibili e quasi impossibili da riesaminare. Fortunatamente, le ultime versioni di Microsoft QuickBASIC hanno portato il linguaggio BASIC nel mondo moderno della programmazione strutturata. GOTO ora è uno strumento antiquato, sostituito da una varietà di strutture di controllo, come i cicli ed i selettori. Inoltre, i programmatori QuickBASIC possono lasciar perdere l'istruzione GOSUB sostituendola con le chiamate a procedure e funzioni. La Parte II descrive gli elementi essenziali della programmazione strutturata in QuickBASIC. Nei capitoli a ciò dedicati, si potrà imparare ad usare i seguenti strumenti:

- Procedure e funzioni, caratterizzate da variabili locali, nonché da argomenti che possono essere passati per indirizzo o per valore.
- I cicli DO con le clausole WHILE o UNTIL poste all'inizio o alla fine della struttura e i cicli FOR controllati da variabili di conteggio che appartengono ai tipi di dati numerici di QuickBASIC.
- Il selettore semplice IF con le clausole opzionali ELSEIF, per esprimere alternative complesse; ed il selettore multiplo SELECT CASE, per selezionare in modo efficiente uno tra diversi blocchi alternativi di codice.

Capitolo 5

Commenti e metacomandi

Un commento è costituito da una o più righe di testo, memorizzate all'interno del listato di un programma, per spiegare il compito ed il comportamento di un particolare blocco di codice. Un programma ben commentato fornisce molta documentazione, scritta per le persone che alla fine lo useranno, lo revisioneranno o semplicemente proveranno a capirne il funzionamento. Un livello appropriato di commento del codice sorgente è un elemento importante di ogni progetto di programmazione, qualunque sia la sua lunghezza. Purtroppo, molti programmatori rinviano la scrittura dei commenti alla fine del progetto, quando rimane poco tempo per eseguire bene il lavoro. Il momento migliore, però, per scrivere i commenti è prima o durante il processo di sviluppo di un programma. La trattazione che segue la programmazione strutturata inizia con una discussione sulle tecniche di commento disponibili in QuickBASIC. È piuttosto interessante sottolineare che l'istruzione REM di QuickBASIC fornisce il contesto sintattico per tre speciali direttive del compilatore chiamate *metacomandi*. Questo capitolo descrive anche queste direttive.

REM

REM designa una riga di commento nel listato di un programma di QuickBASIC. In alternativa, si può usare, come delimitatore di un commento, semplicemente una sola virgoletta ('). Il compilatore QuickBASIC ignora le righe di commento.

```
REM testo di comm
```

o

```
' testo di comm
```

Nei commenti che occupano un'intera riga, la parola chiave REM o la sola virgoletta è all'inizio della riga. In un programma QuickBASIC, si possono mettere i commenti anche alla fine di una riga d'istruzione. Se si usa REM per indicare un commento posto a fine riga, bisogna separare l'istruzione da REM con due punti:

```
ISTRUZIONE : REM testo di comm
```

Al contrario, una sola virgoletta usata come delimitatore di un commento non richiede i due punti:

```
ISTRUZIONE ' testo di comm
```

I commenti sono parte del codice sorgente di un programma QuickBASIC, cioè del listato del programma; non sono inclusi però nel codice compilato risultante.

Alcuni esempi di commento

Ogni programmatore sviluppa uno stile individuale di commento ai programmi. Ecco alcuni suggerimenti pratici, accompagnati da esempi presi dal programma *Mese* (presentato nel Capitolo 29): all'inizio del programma si mette un commento che identifica il nome ed il compito del programma:

```
' MESE.BAS
' Visualizza uno o più mesi del calendario.
```

Altre informazioni possono apparire all'inizio del listato, compresi il nome del programmatore, le date delle revisioni e le note sull'utilizzo. Inoltre, vi è un commento all'inizio di ogni sottoprogramma o funzione, per spiegare cosa esegue la procedura; ad esempio:

```
SUB VisualMese (mese%, anno%)

' La procedura VisualMese visualizza il mese del calendario
' specificato negli argomenti.
```

Si usino brevi commenti per descrivere il compito di blocchi di codice particolarmente significativi all'interno del proprio programma:

```
' Stabilisce se si può stampare.

IF RIGHT$(comriga$, 2) = "/P" THEN
   Output% = Output + stampaInf%
END IF
```

Infine, i commenti linea per linea possono rivelarsi molto utili per descrivere l'uso di variabili e costanti simboliche; ad esempio:

```
CONST commErr% = 0      ' La stringa COMMAND$ dell'utente
                        ' non è valida.
CONST nessComm% = 1     ' L'utente non fornisce nessuna stringa
                        ' COMMAND$.
CONST unMese% = 2       ' L'utente richiede un mese specifico.
CONST meseAtt% = 3      ' La richiesta è per un mese di quest'anno.
CONST annoInt% = 4      ' La richiesta è per un intero anno.
CONST stampaInf% = 5    ' Valore aggiuntivo da stampare.
```

Compatibilità

I commenti hanno la stessa forma sia in Turbo Basic sia in BASICA e si impostano come in QuickBASIC.

L'ISTRUZIONE REM E I METACOMANDI

QuickBASIC ha tre direttive, chiamate *metacomandi*, che eseguono operazioni specifiche nel momento in cui un programma viene compilato. In sintesi, ognuno di questi comandi inizia con il carattere dollaro e deve essere all'interno di un commento. (Questa è l'unica eccezione alla regola per cui il compilatore ignora i commenti.) I metacomandi di QuickBASIC sono i seguenti:

$STATIC e $DYNAMIC, che controllano il modo in cui QuickBASIC alloca la memoria per memorizzare i vettori. Il metacomando $STATIC specifica che una quantità fissa di spazio di memoria sarà allocata per i vettori al momento della compilazione. Al contrario, $DYNAMIC fa sì che QuickBASIC allochi una quantità variabile di spazio per i vettori "dinamicamente", durante l'effettiva esecuzione del programma. (Si possono avere altre notizie su $STATIC e $DYNAMIC consultando il Capitolo 3, "Vettori e record".) $INCLUDE che indirizza il compilatore a leggere un aggiuntivo file del codice sorgente e lo incorpora nella compilazione corrente. La sintassi dei metacomandi $STATIC e $DYNAMIC è semplice, perché nessuna direttiva richiede un argomento. Si possono scrivere questi comandi nel modo seguente:

```
REM $STATIC
REM $DYNAMIC
```

o

```
' $STATIC
' $DYNAMIC
```

Si noti che è possibile usare sia la parola chiave REM sia una sola virgoletta come delimitatore del commento che contiene il metacomando. La sintassi della direttiva $INCLUDE è leggermente più dettagliata. Con questo metacomando si include il nome di un file che comprende il codice sorgente che si vuole incorporare nel proprio programma. Il nome del file è tra due singole virgolette e due punti separano il nome del file dalla direttiva stessa:

```
REM $INCLUDE: 'nomeFile'
```

o

```
' $INCLUDE: 'nomeFile'
```

Un "include file" deve essere disponibile su disco nel formato di testo ASCII o il compilatore QuickBASIC non potrà leggerlo. Gli "include file" dei programmi QuickBASIC 4.0 e 4.5 non possono contenere sottoprogrammi o funzioni. Nelle prime versioni di QuickBASIC (le versioni precedenti a 4.0) il metacomando $INCLUDE era il principale strumento disponibile per

incorporare librerie di procedure già testate in un programma al momento della compilazione. Il nuovo ambiente QuickBASIC (versioni 4.0 e 4.5) offre, però, una tecnica migliore per costruire i programmi: si possono caricare in memoria più *moduli* del programma alla volta, combinando diversi file del codice sorgente in un solo programma. Per questo motivo, l'utilizzo del metacomando $INCLUDE è molto antiquato rispetto alle ultime versioni di QuickBASIC.

Capitolo 6

Sottoprogrammi e funzioni

Una procedura è una porzione del codice che esegue un compito specifico. QuickBASIC 4.5 supporta due tipi di procedure chiamate *sottoprogrammi* e *funzioni*. L'ambiente di sviluppo di QuickBASIC ha molte importanti caratteristiche che semplificano il lavoro con le procedure. Ad esempio, ogni singola procedura viene visualizzata all'interno della propria finestra nell'editor di QuickBASIC. Si possono usare i comandi SUB nel menù View per selezionare ogni procedura esistente, per visualizzare o eseguire modifiche. Un sottoprogramma è una procedura che viene definita in una struttura SUB...END SUB. Una *chiamata* ad un sottoprogramma è un'istruzione che richiede una singola esecuzione della procedura. La definizione del sottoprogramma può comprendere una lista opzionale di variabili, con il compito di ricevere i valori dei parametri trasmessi da una chiamata della procedura. La lista di variabili nell'istruzione SUB è detta *lista dei parametri* e la corrispondente lista dei valori da trasmettere, che compare nell'istruzione di chiamata, è la *lista degli argomenti*. Un'istruzione di chiamata può inviare delle variabili come argomenti in due diversi modi, *per indirizzo*, consentendo alla procedura di cambiare i valori memorizzati nelle variabili specificate come argomenti; o *per valore*, proteggendo gli argomenti da eventuali cambiamenti. Una funzione, definita in una struttura FUNCTION...END FUNCTION, ha molte caratteristiche in comune con un sottoprogramma. La principale differenza è che una funzione viene solitamente progettata per calcolare e ritornare un solo valore al programma chiamante. Per questo motivo, una chiamata ad una funzione non si presenta

Tabella 6-1. Termini corrispondenti in QuickBASIC e BASICA

QuickBASIC	BASICA
procedura (CALL, SUB ... END SUB)	subroutine (GOSUB, RETURN)
funzione (FUNCTION ... END FUNCTION)	funzione definita dall'utente (DEF FN)
dichiarazione di procedura (DECLARE)	(non prevista)

mai come un'istruzione a se stante, ma sempre nell'ambito di un'espressione che fornisce un particolare valore a un'espressione o a un'istruzione più complesse. Quando si scrive e si salva un programma strutturato contenente sottoprogrammi e funzioni, l'editor QuickBASIC automaticamente genera delle dichiarazioni speciali delle procedure, usando la parola chiave DECLARE. Queste istruzioni aiutano il compilatore ad eseguire alcuni compiti di verifica associati alle procedure e alle relative chiamate. Le istruzioni e le strutture che definiscono, chiamano e dichiarano le procedure di QuickBASIC sono argomento di questo capitolo. Le strutture equivalenti di stile BASICA vengono intenzionalmente omesse qui; sono trattate invece nel Capitolo 8, "Altre istruzioni di controllo". Studiando le voci di questo capitolo e del Capitolo 8, si inizierà ad abituarsi alla terminologia che distingue le strutture di QuickBASIC dalle istruzioni più vecchie dello stile BASICA. La Tabella 6-1 riassume questi termini. Esclusi i programmi che richiedono compatibilità con alcune versioni precedenti del linguaggio BASIC, o in alcune istruzioni di gestione degli errori o degli eventi, le strutture in stile BASICA dovrebbero generalmente essere abbandonate in favore di quelle del QuickBASIC.

CALL

Un'istruzione CALL passa il controllo del flusso esecutivo ad un sottoprogramma di QuickBASIC. L'istruzione comprende una lista di argomenti, corrispondente alla lista dei parametri nell'istruzione SUB che definisce il sottoprogramma. QuickBASIC supporta due diversi formati per le chiamate dei sottoprogrammi, con o senza la parola chiave CALL:

```
CALL NomeSot(arg)
```

o

```
NomeSot arg
```

Perché si possa usare il secondo di questi due formati, senza CALL, è richiesto che il programma principale comprenda un'istruzione DECLARE che si riferisce al sottoprogramma. (QuickBASIC automaticamente genera questa dichiarazione quando si salva su disco il programma.) Una chiamata ad un sottoprogramma che non contiene nessuna lista di parametri si presenta semplicemente con lo stesso nome del sottoprogramma, non seguito dalle parentesi:

```
NomeSot
```

D'altra parte, la lista degli argomenti di una chiamata del sottoprogramma deve contenere lo stesso numero di variabili che si trovano nella corrispondente lista dei parametri SUB; ogni argomento deve appartenere ad un tipo che è compatibile con quello della variabile corrispondente nella lista dei parametri. Si può esprimere un argomento come una costante, una variabile o un'espressione. Per default, gli argomenti che sono espressi come variabili, vengono passati al sottoprogramma *per indirizzo*. Ciò significa che QuickBASIC passa l'indirizzo di memoria della variabile stessa al sottoprogramma, piuttosto che una copia del valore che la variabile contiene. Se la routine cambia il valore del parametro corrispondente, la variabile viene "ritrovata" cambiata dalla procedura che chiamò la funzione. Nella struttura di un programma, ciò può essere o un grosso vantaggio, consentendo ad un sottoprogramma di cambiare diversi valori di dati del proprio chiamante, attraverso le variabili passate per valore, o un grave inconveniente, quando un sottoprogramma esegue un cambiamento inaspettato e indesiderato di un valore che il chiamante non voleva variare. Per proteggere una variabile dai cambiamenti, la si può mettere tra parentesi all'interno della lista degli argomenti. Questa speciale notazione significa che l'argomento verrà inviato *per valore* piuttosto che per indirizzo. Ad esempio, la variabile *var2* nella seguente chiamata di funzione viene inviata per valore, mentre *var1* e *var3* vengono inviate per indirizzo:

```
NomeSot var1, (var2), var3
```

Quando si invia un argomento per valore, QuickBASIC invia una copia del valore corrente della variabile, piuttosto che l'indirizzo della variabile. Ogni volta che si vuole essere sicuri che un argomento non sarà cambiato dalla funzione che si sta chiamando, bisogna inviare l'argomento per valore. Le opzioni di passaggio per valore o per indirizzo sono disponibili solo quando gli argomenti vengono espressi come semplici nomi di variabile. Gli argomenti espressi come costanti o nell'ambito di espressioni vengono sempre inviati per valore. Al contrario, i vettori che compaiono come argomenti vengono sempre inviati per indirizzo. In una lista degli argomenti il nome di un vettore è seguito da due parentesi vuote:

```
NomeSot nomeVett()
```

Non si può passare un intero vettore per valore, sebbene si possa passare un solo elemento del vettore per valore. (Vedere il Capitolo 3, "Vettori e record", per ulteriori informazioni.) Infine, un sottoprogramma QuickBASIC può chiamare se stesso, con un processo chiamato *ricorsione*. Una chiamata ricorsiva è una chiamata ad un sottoprogramma all'interno della stessa definizione del sottoprogramma. Alcuni algoritmi, come certe tecniche di ordinamento, si prestano particolarmente bene alle tecniche di ricorsione.

Alcuni esempi di chiamate di sottoprogrammi

I seguenti esempi di chiamate di sottoprogrammi sono prese dal programma *Mese* presentato nel Capitolo 29. Il primo esempio è una chiamata ad una funzione che non richiede argomenti:

```
Output
```

In questo caso, l'istruzione di chiamata è costituita semplicemente dal nome del sottoprogramma. Con la parola chiave CALL, l'istruzione si presenterà così:

```
CALL Output
```

Il secondo esempio indica una chiamata che passa due argomenti ad un sottoprogramma chiamato *VisualMese*:

```
VisualMese me%, an%
```

La routine *VisualMese* legge questi due argomenti interi come il mese e l'anno da cui generare il calendario, che la routine poi visualizza sullo schermo. Con CALL, quest'istruzione apparirà nel seguente modo:

```
CALL VisualMese(me%, an%)
```

Infine, l'istruzione successiva è un esempio di una chiamata che utilizza i dati restituiti da un sottoprogramma:

```
LeggiLiCom operaz%, me%, an%
```

Il sottoprogramma *LeggiLiCom* legge le informazioni dalla linea di comando che l'utente digita al prompt di sistema. (Vedere la voce COMMAND$ per maggiori dettagli.) La routine ha il compito di ritornare le informazioni alla procedura chiamante, attraverso i parametri *qualeOutput%*, *mese%* e *anno%*:

```
SUB LeggiLiCom (qualeOutput%, mese%, anno%)
```

Quando l'esecuzione di *LeggiLiCom* è completata, questi tre valori interi vengono trascritti, attraverso il meccanismo delle chiamate per indirizzo, nelle variabili degli argomenti *operaz%*, *me%* e *an%*.

Compatibilità

Turbo Basic supporta sottoprogrammi ed istruzioni CALL. Diversamente da QuickBASIC, comunque, una chiamata di sottoprogramma in Turbo Basic deve iniziare sempre con la parola chiave CALL. BASICA supporta solo chiamate di sottoprogrammi, eseguite con l'istruzione GOSUB.

DECLARE

L'istruzione DECLARE consente di dichiarare una procedura QuickBASIC che verrà definita ed usata successivamente nel programma corrente. L'ambiente di sviluppo di QuickBASIC automaticamente genera un'istruzione DECLARE per ogni sottoprogramma o funzione che si scrive nel modulo di un programma dato.

```
DECLARE SUB NomeSot (par)
```

o

```
DECLARE FUNCTION NomeFunz (par)
```

Nell'istruzione DECLARE, la parola chiave SUB viene usata per dichiarare un sottoprogramma, e FUNCTION serve per dichiarare una funzione. Per dichiarare un sottoprogramma o una funzione che non prevedono una lista dei parametri, QuickBASIC inserisce delle parentesi vuote dopo il nome della procedura nell'istruzione DECLARE. La presenza dell'istruzione DECLARE nel proprio programma richiede che il compilatore verifichi che il numero ed il tipo di argomenti in ogni chiamata di procedura corrisponda esattamente alla lista dei parametri nella corrispondente definizione di procedura.

Alcuni esempi dell'istruzione DECLARE

Le seguenti istruzioni DECLARE compaiono nella parte principale del programma *Mese*, presentato nel Capitolo 29:

```
DECLARE SUB Output ()
DECLARE SUB LeggiLiCom (qualeOutput%, mese%, anno%)
DECLARE SUB PrepStamp ()
DECLARE SUB StampaMese (mese%, anno%)
DECLARE SUB StampOVisual (st%, me%, an%)
DECLARE SUB VisualMese (mese%, anno%)
DECLARE FUNCTION NumGior% (mese%, giorno%, anno%)
DECLARE FUNCTION NumData& (mese%, giorno%, anno%)
DECLARE FUNCTION AnnoBisest% (anno%)
DECLARE FUNCTION Mese$ (numMese%)
DECLARE FUNCTION GiorniMese% (mese%, anno%)
```

Come si può vedere, il programma *Mese* contiene sei sottoprogrammi e cinque funzioni. Oltre al suo ruolo nella compilazione, una sequenza di istruzioni DECLARE fornisce un riassunto dei moduli contenuti nel proprio programma e serve come guida di riferimento per i nomi e gli argomenti richiesti per le chiamate di sottoprogrammi e funzioni.

Compatibilità

Il Turbo Basic non utilizza le istruzioni DECLARE. Il linguaggio interpretato BASICA non supporta né i sottoprogrammi né le funzioni, e quindi nemmeno l'istruzione DECLARE.

FUNCTION...END FUNCTION

La struttura FUNCTION...END FUNCTION definisce una funzione. In generale, lo scopo di una funzione è produrre e ritornare un solo valore del tipo specificato. Una chiamata di funzione, rappresentata semplicemente dal nome della funzione stessa, può comparire in ogni espressione QuickBASIC che usa appropriatamente il valore ritornato dalla funzione.

```
FUNCTION NomeFunz (par) STATIC ' (par) e STATIC sono facoltativi.
   ' blocco di istruzioni QuickBASIC

   NomFunz = espress

END FUNCTION
```

Le istruzioni FUNCTION e END FUNCTION segnano rispettivamente l'inizio e la fine della definizione della funzione. Tra FUNCTION e END FUNCTION si può inserire un blocco di istruzioni QuickBASIC. Tutte le variabili che si utilizzano all'interno della funzione sono, per default, locali, cioè visibili solo nella funzione. La parola chiave opzionale STATIC specifica che i valori delle variabili locali saranno conservati da una chiamata di funzione all'altra. (Per ulteriori informazioni vedere il paragrafo "Campo di visibilità" nel Capitolo 2.) L'istruzione FUNCTION definisce il nome della funzione e, implicitamente, il tipo di valore che la funzione ritornerà. Il nome di una funzione può essere formato al massimo da 40

caratteri, inizia con una lettera, ma può contenere anche cifre. Il carattere finale indica il tipo di valore che la funzione ritornerà: & per interi lunghi, % per gli interi, # valori in virgola mobile a precisione doppia, ! (o nessun tipo speciale di carattere) per variabili a virgola mobile in precisione semplice e $ per una stringa. Una funzione QuickBASIC non può ritornare una stringa a lunghezza fissa, un vettore o un tipo record definito dall'utente. (Vedere il Capitolo 1 per ulteriori informazioni sui tipi di dati.) Una lista opzionale di parametri, chiusa tra parentesi, può comparire nell'istruzione FUNCTION. Questi parametri rappresentano i dati che sono passati alla funzione da una chiamata. Gli elementi della lista dei parametri possono essere semplici nomi di variabili con caratteri di tipo (&, %, #, ! o $). Si può utilizzare anche la seguente notazione per specificare il tipo di parametri:

```
nomeVariab AS parchiaveTipo
```

dove le parole chiave di tipo disponibili sono LONG, INTEGER, DOUBLE, SINGLE e STRING. Inoltre, la lista dei parametri può comprendere vettori e variabili record. Il nome di un vettore è seguito nella lista dei parametri da parentesi vuote:

```
nomeVett()
```

Come il nome di una variabile, anche il nome di un vettore può finire con un carattere di tipo che indica il tipo dei valori memorizzati nel vettore. Un vettore specificato come parametro deve avere le stesse dimensioni del vettore indicato nella corrispondente chiamata di funzione. Per definire un parametro record, bisogna assegnare il nome di un tipo record predefinito, con questa notazione:

```
nomeVariab AS tipoRec
```

Allo stesso modo, una funzione può ricevere, come parametro, un vettore di record. Ecco la notazione della lista dei parametri per questa struttura:

```
nomeVett() AS tipoRec
```

(Per ulteriori informazioni sui tipi record definiti dall'utente, vedere la voce TYPE nel Capitolo 3.) Un'istruzione FUNCTION senza una lista dei

parametri definisce una funzione che non riceve nessun argomento. In genere una funzione usa le proprie variabili dei parametri per calcolare un nuovo valore, che viene poi ritornato come valore della funzione. Per specificare il valore di ritorno, un'istruzione all'interno della funzione assegna un valore al nome della funzione:

```
nomeFunz = espress
```

Questa istruzione appare di solito alla fine della funzione, ma effettivamente può essere localizzata in ogni punto all'interno della funzione. QuickBASIC genera automaticamente un'istruzione DECLARE per ogni funzione che si definisce in un programma. Questa istruzione è essenziale per l'uso corretto di una funzione. (Vedere la voce DECLARE per ulteriori dettagli.)

Chiamate di funzione

Una chiamata di funzione è rappresentata dal nome della funzione, seguito da una lista appropriata di argomenti tra parentesi:

```
Nome(arg)
```

Una chiamata di funzione non è un'istruzione in QuickBASIC. Piuttosto, una chiamata è sempre parte di un'espressione o di un'istruzione che usa il valore che la funzione ritorna. Ad esempio, si supponga di definire una funzione nel seguente modo:

```
FUNCTION NomeFunz (var1, var2, var3)
```

La chiamata di funzione corrispondente si presenta così:

```
NomeFunz(val1, val2, val3)
```

La lista degli argomenti nella chiamata di funzione corrisponde alla lista dei parametri nella definizione della funzione. Nella lista di argomenti deve apparire lo stesso numero di valori che ci sono nella lista dei parametri, ed ogni valore degli argomenti deve appartenere ad un tipo che sia compatibile con il corrispondente parametro. Una chiamata ad una funzione che non prevede una lista dei parametri si presenta semplicemente con lo stesso nome della funzione:

```
NomeFunz
```

In questo caso non si devono mettere le parentesi vuote dopo il nome della funzione. Un argomento si può esprimere come una costante, una variabile o come un'espressione. Per default, gli argomenti espressi come variabili vengono passati alla funzione *per indirizzo*. QuickBASIC passa l'indirizzo in memoria della variabile stessa alla funzione, piuttosto che una copia del valore che la variabile contiene. Se la funzione cambia il valore del parametro corrispondente, questo valore modificato viene "trascritto" nella procedura che chiamò la funzione. Nella struttura di un programma, ciò può essere o un grosso vantaggio, consentendo ad una funzione di restituire più valori al proprio chiamante, attraverso i parametri, o un grosso inconveniente, quando una funzione esegue un cambiamento inaspettato e indesiderato in un valore che il chiamante voleva mantenere invariato. Si può proteggere una variabile dai cambiamenti, inserendola tra parentesi all'interno della lista degli argomenti. Questa speciale notazione significa che l'argomento verrà inviato *per valore* piuttosto che per indirizzo. Ad esempio, la variabile *var2* nella seguente chiamata di funzione viene inviata per valore, mentre *var1* e *var*3 vengono inviate per indirizzo:

```
NomeFunz(var1, (var2), var3)
```

Quando si invia un argomento per valore, QuickBASIC manda una copia del valore corrente della variabile, piuttosto che l'indirizzo della variabile stessa. Ogni volta che si vuole proteggere un programma da eventuali cambiamenti del valore di una variabile passata come argomento, bisognerebbe inviare l'argomento per valore. La distinzione tra il passaggio per valore e quello per indirizzo si fa solo per gli argomenti che vengono espressi come semplici nomi di variabile. Gli argomenti espressi come costanti ed espressioni vengono sempre inviati per valore. Al contrario, i vettori specificati come argomenti vengono sempre inviati per indirizzo. Non si può inviare un intero vettore per valore, sebbene si possa invece inviare un singolo elemento del vettore per valore. Una funzione QuickBASIC può chiamare se stessa con un processo chiamato *ricorsione*. Una chiamata ricorsiva è una chiamata ad una funzione all'interno della stessa definizione della funzione.

Alcuni esempi di definizione di funzione e chiamate di funzione

Il programma *Mese* (presentato nel Capitolo 29) contiene molte interessanti definizioni di funzione. Un breve esempio è una funzione chiamata *AnnoBisest%*. Questa funzione riceve un argomento intero, un anno del calendario a quattro cifre. *AnnoBisest%* ritorna un intero che QuickBASIC può leggere come un valore logico, vero se l'argomento è un anno bisestile, falso se non lo è.

```
FUNCTION AnnoBisest% (anno%)

' La funzione AnnoBisest% ritorna un valore booleano che
' indica se un determinato anno è bisestile.

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)

   AnnoBisest% = divPer4% AND (secolo% IMP secolo400%)

END FUNCTION
```

La funzione esegue tre prove sul parametro *anno%* per vedere se rappresenta un anno bisestile. Generalmente, un anno bisestile è un anno perfettamente divisibile per 4, mentre i secoli sono anni bisestili se sono perfettamente divisibili per 400. (Ad esempio, 2000 è un anno bisestile, ma 1900 no.) Per determinare ciò, la funzione *AnnoBisest%* crea tre variabili logiche e poi usa le operazioni AND e IMP per valutare un'espressione finale che diventa il valore di ritorno. (Per ulteriori informazioni consultare il paragrafo "Operazioni logiche" del Capitolo1.) Ecco un esempio di una chiamata della funzione *AnnoBisest%*:

```
FOR annoCorr% = annoIniz% TO anno% - 1 STEP 4
   IF AnnoBisest%(annoCorr%) THEN num& = num& + 1
NEXT annoCorr%
```

Questo ciclo LOOP, che si trova all'interno di una funzione che si chiama *NumData&*, esegue ripetute chiamate ad *AnnoBisest%* nel processo di produzione di una rappresentazione intera per una data del calendario. Per ogni anno che si scopre essere bisestile, la funzione aggiunge 1 a *num&*, il numero

accumulato di giorni da una data fissa d'inizio. Una funzione può contenere una chiamata ad un'altra funzione. Ad esempio, si consideri questo secondo esempio, chiamato *NumData%:*

```
FUNCTION NumGior% (mese%, giorno%, anno%)

'  La funzione NumGior% ritorna un valore da 1 a 7,
'     che rappresenta il giorno della settimana (da
'     domenica a sabato) della data specificata negli
'     argomenti. In un valore di ritorno pari a zero
'     risulta una data non valida.

   g& = numData&(mese%, giorno%, anno%)
   IF g& <> 0 THEN gds% = g& MOD 7 + 1 ELSE gds% = 0

   NumGior% = gds%

END FUNCTION
```

Dati tre argomenti interi che rappresentano una data del calendario, questa funzione ritorna un intero da 1 a 7, che indica il giorno in cui cade la data, da domenica a sabato. Si noti che la funzione *NumGior%* inizia con una chiamata ad un'altra funzione, *NumData&*, che dà l'equivalente intero di una data. Per ulteriori informazioni su queste funzioni, vedere la descrizione del programma *Mese* nel Capitolo 29.

Compatibilità

Le funzioni non sono supportate né in Turbo Basic né in BASICA. La struttura FUNCTION rappresenta perciò uno dei maggiori vantaggi di QuickBASIC su altre versioni del linguaggio. Turbo Basic non consente funzioni multilinee DEF FN, e BASICA supporta una versione su una sola linea di DEF FN, ma la struttura di DEF FN è meno utile e potente di FUNCTION...END FUNCTION. (QuickBASIC ha anche l'istruzione DEF FN, primariamente per compatibilità. Vedere la voce DEF FN per ulteriori dettagli.)

SUB...END SUB

La struttura di SUB...END SUB definisce un sottoprogramma QuickBASIC. Un sottoprogramma esegue un'operazione definita nel contesto di un particolare progetto di programmazione. Una chiamata ad un sottoprogramma, rappresentato dal nome del sottoprogramma stesso, rappresenta un'istruzione a se stante. (Vedere anche la voce CALL.)

```
SUB NomeSot (par) STATIC  ' (par) e STATIC sono opzionali.
 'blocco di istruzioni QuickBASIC

END SUB
```

Le istruzioni SUB e END SUB segnano l'inizio e la fine della definizione di un sottoprogramma. Tra SUB e SUB END si può inserire un blocco di istruzioni QuickBASIC. Tutte le variabili che si utilizzano all'interno del sottoprogramma sono, per default, locali, cioè visibili solo nel sottoprogramma. La parola chiave opzionale STATIC specifica che i valori delle variabili locali saranno conservati da una chiamata del sottoprogramma a quella successiva. (Per ulteriori informazioni vedere il paragrafo "Campo di visibilità" nel Capitolo 2.) L'istruzione SUB definisce il nome del sottoprogramma. Un nome può essere formato al massimo da 40 caratteri, inizia con una lettera, ma può contenere anche cifre. Dopo il nome si può aggiungere una lista facoltativa di parametri, messa tra parentesi. Questi parametri rappresentano i dati che verranno passati al sottoprogramma da una chiamata. Le voci nella lista dei parametri possono essere semplici nomi di variabili con caratteri di tipo (&, %, #, ! o $). Si può utilizzare anche la seguente notazione per specificare il tipo di un parametro:

```
nomeVariab AS parchiaveTipo
```

dove le parole chiave disponibili di tipo sono LONG, INTEGER, DOUBLE, SINGLE e STRING. Inoltre, un sottoprogramma può ricevere come parametri vettori e record. Il nome di un vettore deve essere seguito da parentesi vuote nella lista dei parametri:

```
nomeVett()
```

Come il nome di una variabile, il nome di un vettore può finire con un carattere di tipo che indica il tipo dei valori memorizzati nel vettore. Le dimensioni del vettore non sono specificate nella lista dei parametri. Quando il sottoprogramma viene chiamato, un parametro vettore assume le stesse dimensioni del vettore corrispondente che è inviato al sottoprogramma come un argomento. Per definire un parametro record, bisogna assegnare il nome di un tipo di record predefinito, con questa notazione:

```
nomeVariab AS tipoRecord
```

Allo stesso modo, un sottoprogramma può ricevere un vettore di record; si utilizzi questa notazione nella lista dei parametri:

```
nomeVett() AS tipoRecord
```

(Per ulteriori informazioni sui tipi record definiti dall'utente vedere la voce TYPE nel Capitolo 3.) Un'istruzione SUB con nessuna lista dei parametri definisce un sottoprogramma che non riceve nessun argomento. QuickBASIC genera automaticamente un'istruzione DECLARE per ogni sottoprogramma che si definisce in un programma. (Vedere la voce DECLARE per ulteriori dettagli.) La chiamata di un sottoprogramma viene eseguita dall'istruzione CALL in QuickBASIC, sebbene la stessa parola chiave CALL sia facoltativa. Gli argomenti espressi come nomi di una variabile scalare vengono normalmente passati per indirizzo ad un sottoprogramma, ma un argomento chiuso tra parentesi nell'istruzione CALL viene invece passata per valore. (Vedere la voce CALL per ulteriori dettagli ed un esempio.) Generalmente, QuickBASIC esegue le istruzioni all'interno di un blocco SUB...END SUB in sequenza dall'inizio alla fine; comunque, si può usare il comando EXIT SUB per eseguire un'uscita non strutturata dalla procedura. (Vedere la voce EXIT.)

Un esempio di sottoprogramma

Un sottoprogramma contiene un blocco di istruzioni QuickBASIC eseguibili, che comprendono facoltativamente chiamate *ricorsive* alla procedura stessa e/o chiamate ad altre procedure. Il seguente breve sottoprogramma (un frammento del programma *Mese* presentato nel Capitolo 29)

usa il valore del suo primo argomento intero per decidere tra chiamate a due altri sottoprogrammi. La procedura, chiamata *StampOVisual*, ha il compito di inviare un mese del calendario alla stampante o visualizzarlo sullo schermo:

```
SUB StampOVisual (st%, me%, an%)

' La procedura StampOVisual chiama o la routine
' StampMese per stampare un mese, o la routine
' VisualMese per visualizzare un mese sullo schermo.
' La scelta tra queste due opzioni dipende dal valore
' logico passato a st%.

  IF st% THEN

' Stampa il mese.

   StampMese me%, an%
   LPRINT

ELSE

' Visualizza il mese sullo schermo.

      VisualMese me%, an%
      PRINT

   END IF

END SUB
```

Come si può vedere, il parametro *st%* viene usato come un valore logico nel selettore IF...THEN...ELSE del sottoprogramma. La routine poi chiama o *StampMese* o *VisualMese*, inviando i valori di *me%* e *an%* alla procedura scelta.

Compatibilità

La struttura SUB...END SUB in Turbo Basic è la stessa di quella in QuickBASIC BASICA, però non supporta sottoprogrammi ma solo procedure chiamate con l'istruzione GOSUB.

Capitolo 7

Cicli e selettori

Un ciclo è una struttura che esegue un blocco di istruzioni ripetutamente. Un selettore, invece, determina se eseguire un particolare frammento di codice o selezionare un frammento del codice tra molte opzioni disponibili. QuickBASIC offre molte istruzioni che si possono usare per esprimere cicli e selettori. Questi due tipi essenziali di strutture di controllo sono l'argomento di questo capitolo.

I CICLI

QuickBASIC offre tre strutture per esprimere i cicli: DO...LOOP, FOR...NEXT e WHILE...WEND. In ogni caso, un'istruzione funge da indicatore dell'inizio del ciclo (DO, FOR o WHILE) e un'altra ne indica la fine (LOOP, NEXT o WEND). Il frammento di codice che deve essere ripetuto è racchiuso tra due indicatori. Questo blocco può contenere qualsiasi istruzione QuickBASIC eseguibile, comprese altre strutture ciclo *nidificate*. Includendo cicli all'interno di cicli, si possono controllare complesse azioni ripetitive. Un'esecuzione completa del frammento di codice che si trova all'interno di un ciclo è definita *iterazione*. Ogni tipo di ciclo ha il proprio sistema per determinare quando le iterazioni devono terminare. Ad esempio, la struttura DO...LOOP usa una condizione logica che controlla quanto dura il ciclo. DO...LOOP è versatile: la condizione può comparire all'interno di una clausola WHILE, nel qual caso il ciclo si blocca quando

un evento commuta il valore della condizione in falso, oppure in una clausola UNTIL, caso in cui le iterazioni si bloccano quando la condizione diventa vera. (Nella struttura WHILE...WEND, che è un precursore BASICA meno flessibile di DO...LOOP, la durata del ciclo è controllata da una condizione WHILE.) Al contrario, l'operazione di un ciclo FOR è controllata da una *variabile di conteggio*. Durante il ciclo questo contatore numerico assume un intervallo progressivo di valori; le iterazioni si bloccano quando la variabile viene *incrementata* o *diminuita* ad un valore che è al di fuori dell'intervallo specificato. Una clausola opzionale STEP determina di quanto deve variare il valore del contatore dopo ogni iterazione. Ogni volta che si usa una delle strutture del ciclo di QuickBASIC ci si può fare una domanda importante: il ciclo risulterà sempre in almeno un'iterazione, senza tener conto delle condizioni iniziali, oppure il ciclo in alcune condizioni non risulterà in nessuna iterazione? La risposta dipende dalla sintassi del ciclo scelta. La struttura FOR...NEXT di solito esegue i cicli che richiedono un numero predeterminato di iterazioni. Comunque, FOR...NEXT non risulta in nessuna iterazione se l'intervallo per il contatore è incompatibile con la variazione indicata da STEP. (Vedere la voce FOR...NEXT per ulteriori dettagli.) Allo stesso modo, un ciclo WHILE...WEND non effettua nessuna iterazione se la condizione di WHILE è falsa all'inizio. La sintassi della struttura DO...LOOP dà un controllo esplicito sul fatto che vada comunque eseguita la prima iterazione o meno. Si può infatti collocare la condizione WHILE o UNTIL all'inizio o alla fine del ciclo. Con la clausola situata all'inizio del ciclo, non verrà eseguita nessuna iterazione se la condizione ha il valore iniziale errato. Al contrario, mettere la condizione WHILE o UNTIL alla fine del ciclo assicura che il ciclo eseguirà almeno un'iterazione, senza tener conto del valore iniziale della condizione. Questo capitolo esamina le strutture ciclo di QuickBASIC e per ciascuna di esse spiega in dettaglio la sintassi che controlla il ciclo.

DO...LOOP

La struttura DO...LOOP rappresenta un'operazione ripetitiva in un programma QuickBASIC. Si può usare questa struttura per eseguire un frammento di codice tutte le volte che si vuole e per esprimere una condizione che determina quando il ciclo deve terminare.

```
DO    ' Con una condizione facoltativa WHILE o UNTIL.
   ' frammento di codice QuickBASIC
LOOP  ' Con una condizione facoltativa WHILE o UNTIL.
```

L'istruzione DO contrassegna sempre l'inizio di una struttura DO...LOOP e l'istruzione LOOP ne contrassegna la fine. Qualsiasi numero di istruzioni QuickBASIC può comparire all'interno del ciclo, compresi cicli nidificati e altre strutture nidificate. Per ogni iterazione del ciclo, le istruzioni all'interno del ciclo vengono eseguite dall'inizio alla fine. Una clausola WHILE o UNTIL può comparire all'inizio del ciclo, come parte dell'istruzione DO, oppure alla fine del ciclo, nell'istruzione LOOP. Il formato di queste condizioni è semplice.

```
WHILE condizione
```

o

```
UNTIL condizione
```

dove la *condizione* è un qualsiasi valore, variabile o espressione che QuickBASIC può valutare come vero o falso. (Per ulteriori dettagli consultare i paragrafi sulle operazioni relazionali e logiche nel Capitolo 1.) Questa condizione in definitiva determina quando il ciclo deve terminare: una clausola WHILE nel ciclo specifica che le iterazioni continueranno finché la condizione è vera. Una clausola UNTIL consente al ciclo di continuare fintanto che la condizione è falsa e lo ferma quando la condizione diventa vera. Collocando la condizione WHILE o UNTIL all'inizio o alla fine del ciclo si specifica se almeno un'iterazione è garantita. Una condizione all'inizio del ciclo si presenta così:

```
DO WHILE condizione
```

o

```
DO UNTIL condizione
```

Con la condizione collocata all'inizio del ciclo, il ciclo non eseguirà nessuna iterazione se la condizione non è rispettata fin da principio, cioè se la

condizione WHILE è falsa o se la condizione UNTIL è vera. Una condizione alla fine del ciclo si presenta così:

```
LOOP WHILE condizione
```

o

```
LOOP UNTIL condizione
```

Quando la condizione si trova alla fine del ciclo (nell'istruzione LOOP), il ciclo scandisce almeno un'iterazione, senza tener conto del valore iniziale dell'espressione condizionante. Dopo questa prima iterazione, QuickBASIC valuta la condizione per determinare se continuare o meno. Si può scrivere anche un ciclo che non contiene nessuna condizione WHILE o UNTIL:

```
DO
   ' istruzioni
LOOP
```

In questo formato atipico, la lunghezza del ciclo non è controllata dalla struttura stessa del ciclo. Piuttosto, alcuni eventi all'interno del ciclo possono alla fine essere responsabili della fermata delle iterazioni. Ad esempio, la struttura può contenere un'istruzione EXIT DO che determina il ciclo. (Vedere la voce EXIT per ulteriori dettagli.)

Alcuni esempi di DO...LOOP

Questo breve esempio, preso dal programma *Mese* presentato nel Capitolo 29, illustra l'uso di una condizione WHILE; lo scopo del ciclo è creare una pausa nel programma finché l'utente non preme la barra spaziatrice:

```
DO WHILE barSp$ <> " "
   barSp$ = INKEY$
LOOP
```

Il ciclo continua finché la funzione INKEY$ ritorna un carattere di spaziatura. Il seguente ciclo dà lo stesso risultato:

```
   barSp$ = INKEY$
LOOP UNTIL barSp$ = " "
```

Supponendo che la variabile *barSp$* sia inizializzata come una stringa vuota prima che il ciclo inizi, la posizione della clausola WHILE o UNTIL non è di nessuna importanza in questo semplice esempio. Almeno un'iterazione del ciclo verrà sempre eseguita. Come altro esempio, si consideri il seguente ciclo, preso anch'esso dal programma *Mese*:

```
DO
   FOR giorCorr% = 1 TO 7
      PrimGior% = (giorCorr% AND dataCorr% = 0)
      NelMese% = (dataCorr% > 0 AND dataCorr% < lunghMese%)
      IF PrimGior% OR NelMese% THEN
         dataCorr% = dataCorr% + 1
         PRINT USING " ##"; dataCorr%;
      ELSE
         PRINT " ";
      END IF
   NEXT giorCorr%
   PRINT
LOOP UNTIL dataCorr% = lunghMese%
```

Il ciclo comporta una visualizzazione dei giorni e delle settimane in un dato mese. La struttura esterna DO...LOOP ripete l'elaborazione fin che l'ultima data emessa *(dataCorr%)* è uguale al numero predeterminato di giorni *(lunghMese%)* nel mese specificato. La struttura contiene un ciclo FOR nidificato che scandisce i sette giorni di ogni settimana ed una struttura nidificata IF che determina se una data assegnata fa parte effettivamente del mese. In QuickBASIC queste strutture nidificate illustrano la potenziale complessità di cicli e selettori. (Per degli esempi dell'output effettivo da questo ciclo, vedere la descrizione del programma *Mese* nel Capitolo 29.)

Compatibilità

Turbo Basic fornisce una struttura DO...LOOP che è identica a quella di QuickBASIC. BASICA ha i cicli WHILE...WEND e FOR...NEXT (i quali entrambi sono disponibili in QuickBASIC), ma non la struttura DO...LOOP.

FOR...NEXT

La struttura FOR...NEXT esegue un'operazione ripetitiva controllata da un contatore. Durante il ciclo, il contatore si incrementa assumendo dei valori in un dato intervallo ed un'iterazione viene eseguita per ogni valore nell'intervallo.

```
FOR var = val1 TO val2 STEP interv    ' la clausola STEP
                                      ' è opzionale.
   ' un frammento di codice QuickBASIC

NEXT var
```

L'istruzione FOR contrassegna sempre l'inizio di questa struttura e l'istruzione NEXT la fine. Il contatore del ciclo può appartenere ad un tipo qualsiasi di dati numerici disponibili in QuickBASIC, compresi gli interi, gli interi lunghi, i dati a virgola mobile in precisione semplice e doppia. Il valore iniziale per questa variabile è *val1*, il valore finale è *val2* ed il valore viene cambiato dopo ogni iterazione di un valore pari al passo STEP. Si possono esprimere *val1*, *val2* ed *interv* come costanti numeriche, variabili o espressioni. Il modo di funzionamento del ciclo FOR...NEXT è il seguente: per la prima iterazione del ciclo, il contatore viene inizializzato come *val1*. Dopo ogni iterazione, il contatore viene incrementato dal valore STEP (o diminuito, se il valore STEP è negativo.) QuickBASIC poi confronta il nuovo valore del contatore con *val2*. Se il contatore non è ancora passato a *val2*, viene eseguita un'altra iterazione; altrimenti il ciclo si ferma ed il controllo passa all'istruzione successiva nel programma. Se si omette la clausola opzionale STEP, QuickBASIC incrementa il contatore di 1 dopo ogni iterazione. Il valore STEP può essere negativo e/o frazionario. Le seguenti due condizioni risultano in un ciclo FOR che non esegue iterazioni: il valore iniziale (*val1*) è maggiore del valore finale (*val2*) ed il valore STEP è positivo. Il valore iniziale (*val1*) è minore del valore finale (*val2*) ed il valore STEP è negativo. Se una di queste due condizioni è vera, QuickBASIC salta l'intero blocco FOR...NEXT ed invia il controllo del programma alla prima istruzione dopo NEXT. Un ciclo FOR, in cui *val1* e *val2* sono uguali, risulta sempre esattamente in un'iterazione. All'interno del ciclo può figurare un numero qualsiasi di istruzioni QuickBASIC, compresi cicli nidificati ed altre strutture nidificate. Per ogni iterazione di un ciclo, le istruzioni

all'interno del ciclo vengono eseguite dall'inizio alla fine. Le istruzioni possono usare, e di solito lo fanno, il valore del contatore nell'eseguire particolari compiti attinenti all'elaborazione ripetitiva. Ad esempio, il contatore potrebbe presentarsi come un indice in uno o più vettori all'interno del ciclo, consentendo di elaborare efficientemente un intervallo dei valori del vettore. Utilizzare un'istruzione all'interno del ciclo per cambiare il valore del contatore può portare ad un comportamento imprevisto, e ciò dovrebbe essere evitato. Una variazione possibile nella sintassi della struttura FOR...NEXT consente di usare una sola istruzione NEXT per contrassegnare la fine di più cicli FOR nidificati; ad esempio:

```
NEXT var3, var2, var1
```

Tuttavia, per chiarezza è meglio evitare questo cambiamento ed invece scrivere una sola istruzione NEXT per contrassegnare la fine di ogni ciclo FOR nidificato.

Alcuni esempi di FOR...NEXT

Questi esempi del ciclo FOR sono presi dal programma *Mese*, presentato nel Capitolo 29. Il primo esempio mostra un semplice ciclo FOR in cui i valori iniziali e finali del contatore vengono espressi come costanti:

```
FOR mese% = 1 TO 12
   StampOVisual Output%, me%, anno%
NEXT mese%
```

Il ciclo esegue dodici chiamate ad una procedura che si chiama *StampOVisual*, visualizzando (o stampando) tutti i dodici mesi di uno specifico anno del calendario. Si noti che, in questo ciclo FOR, non c'è nessuna clausola STEP; il contatore *mese%* viene incrementato di 1 alla fine di ogni iterazione. Nei due esempi seguenti, i valori iniziali e finali per i contatori sono espressi come variabili o espressioni:

```
FOR annoCorr% = annoIniz% TO anno% - 1 STEP 4
   IF AnnoBisest%(annoCorr%) THEN num% + 1
NEXT annoCorr%
```

```
FOR meseCorr% = Genn% TO mese% - 1
   num& = num& + GiorniMese%(meseCorr%, anno%)
NEXT meseCorr%
```

Questi due cicli fanno parte di una procedura che trasforma una data del calendario in un valore rappresentativo di tipo intero lungo. Il primo dei due cicli aggiunge giorni extra all'intero per gli anni bisestili; si noti la clausola STEP, che specifica un valore d'incremento pari a 4. L'esempio finale mostra un ciclo FOR che è nidificato all'interno del ciclo DO:

```
DO
   FOR giorCorr% = 1 TO 7
      PrimGior% = (dataCorr% = primGior% AND dataCorr% = 0)
      NelMese% 0 = (dataCorr% > 0 AND dataCorr% < lunghMese%)
      IF PrimGior% OR NelMese% THEN
         dataCorr% = dataCorr% + 1
         PRINT USING " ##"; dataCorr%;
      ELSE
         PRINT "  ";
      END IF
   NEXT giorCorr%
   PRINT
LOOP UNTIL dataCorr% = lunghMese%
```

Questo gruppo di strutture nidificate sono responsabili della visualizzazione di un mese dato del calendario. Il ciclo interno FOR visualizza ogni settimana del mese ed il ciclo esterno DO continua l'elaborazione fino a che non viene visualizzato l'intero mese. Per esaminare questi cicli nel contesto, si osservi la descrizione del programma *Mese* nel Capitolo 29.

Compatibilità

Il ciclo FOR...NEXT di Turbo Basic ha le stesse proprietà e caratteristiche di QuickBASIC. In BASICA, però, c'è un'importante limitazione: il contatore può essere solo di tipo intero o in virgola mobile a precisione semplice.

WHILE...WEND

La struttura WHILE...WEND controlla un'operazione ripetitiva in un programma QuickBASIC. WHILE...WEND è meno versatile della struttura

DO...LOOP, che è la struttura preferibile da usare nei programmi QuickBASIC. Cionondimeno, WHILE...WEND fornisce compatibilità e continuità con versioni più vecchie del linguaggio BASIC.

```
WHILE condizione

   ' un blocco di istruzioni QuickBASIC

WEND
```

L'istruzione WHILE contrassegna sempre l'inizio di una struttura WHILE...WEND e l'istruzione WEND ne contrassegna la fine. Qualsiasi numero di istruzioni QuickBASIC può comparire all'interno del ciclo, compresi i cicli *nidificati* e altre strutture nidificate. Per ogni *iterazione* del ciclo, le istruzioni all'interno del ciclo vengono eseguite dall'inizio alla fine.

```
WHILE condizione
```

o

```
UNTIL condizione
```

dove la condizione nell'istruzione WHILE è un valore qualsiasi, variabile o espressione che QuickBASIC può valutare come vero o falso. (Per ulteriori dettagli consultare i paragrafi sulle operazioni relazionali e logiche nel Capitolo 1.) Le iterazioni del ciclo continuano fino a quando questa condizione è vera. Il ciclo termina quando la condizione WHILE diventa falsa. Se la condizione è falsa all'inizio, non viene eseguita nessuna iterazione.

Un esempio di WHILE...WEND

Il seguente ciclo è una versione WHILE...WEND di un esempio presentato nel paragrafo DO...LOOP.

```
WHILE barSp% <> " "
   barSp$ = INKEY$
WEND
```

Il compito di questo ciclo è creare una pausa nell'esecuzione di un programma fino a che l'utente preme la barra spaziatrice sulla tastiera.

Compatibilità

Sia Turbo Basic sia BASICA forniscono la struttura WHILE...WEND.

I SELETTORI

QuickBASIC supporta due versatili strutture di selettori: IF...THEN...ELSE e SELECT CASE. Ognuna di queste strutture delimita uno o più frammenti di codice, ognuno dei quali può essere selezionato per l'esecuzione. Nella struttura IF...THEN...ELSE, una condizione o una serie di condizioni determina quale blocco di codice, se ce ne sono, verrà selezionato. Ogni frammento può contenere varie istruzioni QuickBASIC, comprese le strutture *nidificate*. Includendo una struttura del selettore in un'altra, si possono ottenere molti selettori complessi. Per selettori più semplici, si può scegliere la versione che occupa una sola linea dell'istruzione IF, che sceglie semplicemente di eseguire una o due istruzioni. La struttura SELECT CASE seleziona un frammento di codice da una serie di blocchi. La tecnica per decidere quale frammento eseguire implica dei confronti tra un valore principale SELECT CASE ed una serie di espressioni CASE. Quando si trova un'uguaglianza, il corrispondente blocco CASE viene eseguito e poi il controllo esce dalla struttura SELECT CASE. Le strutture IF...THEN...ELSE e SELECT CASE vengono esaminate dettagliatamente in questi paragrafi. (Per compatibilità con BASICA, QuickBASIC supporta anche selettori multipli che sono equivalenti a SELECT CASE:ON...GOTO e ON...GOSUB. Queste istruzioni vengono discusse nel Capitolo 8.)

IF...THEN...ELSE

La struttura IF...THEN...ELSE consente di organizzare azioni alternative in frammenti separati di codice. Quello effettivamente eseguito dipende dal valore logico di una o più condizioni espresse nella struttura. QuickBASIC supporta anche una versione abbreviata che occupa una sola linea dell'istruzione IF...THEN...ELSE.

Formato strutturato a blocchi:

```
IF condizione THEN
   ' blocco di istruzioni

ELSEIF condizione            ' Le clausole ELSEIF sono opzionali.
   ' blocco di istruzioni

ELSEIF condizione            ' Sono ammessi più blocchi ELSEIF.
   ' blocco di istruzioni

ELSE condizione              ' La clausola ELSE è opzionale.
   ' blocco di istruzioni

END IF
```

Formato su una sola linea:

```
IF condizione THEN istruzione1 ELSE istruzione2    ' La clausola ELSE
                                                   ' è facoltativa.
```

In entrambi i formati, una *condizione* è qualsiasi espressione che QuickBASIC può valutare vera o falsa. Si possono usare operatori relazionali e/o logici per costruire questa espressione o si può fornire un valore numerico. (Un valore pari a zero è equivalente a falso, e un valore diverso da zero è vero. Per ulteriori dettagli vedere il paragrafo "Operazioni relazionali e logiche" nel Capitolo 1.) Il risultato di un'istruzione IF strutturata a blocchi dipende dal fatto che siano state incluse le clausole opzionali ELSEIF e ELSE. Nella struttura sono ammesse più clausole ELSEIF. QuickBASIC inizia valutando la condizione IF. Se questa prima condizione è vera, il blocco di istruzioni collocato subito dopo l'istruzione IF viene eseguito, e poi il controllo passa alla prima istruzione dopo END IF. Diversamente, QuickBASIC salta alla prima clausola ELSEIF e valuta la sua condizione. Se la condizione è falsa, QuickBASIC continua a saltare alla condizione ELSEIF successiva. Solo quando una data condizione si rivela vera viene eseguito il blocco corrispondente di istruzioni e poi il controllo passa all'istruzione dopo END IF. Nel caso in cui la condizione IF e tutte le condizioni ELSEIF siano false, QuickBASIC salta alla clausola ELSE ed esegue il frammento di istruzioni che si trova tra ELSE e END IF. Senza una clausola ELSE, se tutte le condizioni sono false la struttura del selettore non esegue nessuna operazione. Una struttura del selettore che contiene solo

blocchi IF e ELSE è più semplice: la scelta avviene tra due frammenti di codice. Nel caso in cui la condizione IF sia vera, QuickBASIC esegue tutto il blocco di istruzioni che si trova tra IF e ELSE. Se la condizione è falsa, viene invece eseguito il blocco collocato tra ELSE e END IF. Il frammento che si trova dopo una clausola IF, ELSEIF o ELSE può includere ogni istruzione eseguibile di QuickBASIC, compresi i selettori nidificati o i cicli nidificati. Il formato dell'istruzione IF che occupa una sola linea sceglie tra due istruzioni da eseguire. Se la condizione è vera, viene selezionata l'istruzione collocata dopo THEN; se la condizione è falsa, viene scelta l'istruzione dopo ELSE. In realtà alle clausole THEN e ELSE effettivamente si possono includere più istruzioni, separate dai due punti. Se si omette la clausola opzionale ELSE, se la condizione è falsa l'istruzione non risulta in nessuna operazione. Le clausole THEN e ELSE in un selettore su una sola linea possono comprendere le istruzioni GOTO. D'altra parte, però, la maggior parte dei programmatori QuickBASIC evita l'uso di GOTO.

Alcuni esempi del formato strutturato a blocchi IF

Nel programma *Mese*, presentato nel Capitolo 29, si trovano molte strutture a selettore semplici e complesse. Questi tre esempi illustrano l'utilizzo delle varie versioni della struttura. Il primo esempio, un frammento della funzione chiamata *GiorniMese%*, non contiene nessuna clausola ELSEIF o ELSE. Il suo compito è verificare la validità di un valore dell'argomento, *numMese%* prima della funzione *GiorniMese%* continua ad eseguire il proprio compito di determinare il numero di giorni in un dato mese. Se *numMese%* è minore di 1 o maggiore di 12, un'istruzione EXIT FUNCTION termina la procedura e ritorna il valore zero:

```
IF numMese% < 1 OR numMese% > 12 THEN
   GiorniMese% = 0
   EXIT FUNCTION
END IF
```

D'altra parte, se *numMese%* è un valore compreso nell'intervallo ammesso che va da 1 a 12, questa struttura IF non risulta in nessuna operazione e l'esecuzione della funzione continua. L'esempio successivo è preso da un sottoprogramma chiamato *VisualMese*, il cui compito è visualizzare sullo schermo un mese dato. Nella prima settimana di un mese visualizzato, ci

possono essere molti giorni vuoti sul calendario, a seconda del giorno della settimana in cui il mese inizia. La seguente istruzione IF determina se un dato giorno deve essere visualizzato come un numero, cioè, come parte del mese corrente, o come uno spazio:

```
IF PrimGior% OR NelMese% THEN
   dataCorr% = dataCorr% + 1
   PRINT USING " ##"; dataCorr%;
ELSE
   PRINT "  ";
END IF
```

Si noti che la condizione nell'istruzione IF è costituita da un'espressione OR, che unisce i valori di due variabili intere. Il sottoprogramma assegna i valori logici di queste variabili proprio prima della struttura IF:

```
PrimGior% = (giorCorr% = PrimGior% AND dataCorr% = 0)
NelMese% = (dataCorr% > 0 AND dataCorr% < lunghMese%)
```

Il valore di *PrimGior%* è vero se il giorno corrente è il primo giorno del mese; *NelMese%* è vero fintanto che un data assegnata è ancora all'interno dell'intervallo dei giorni del mese. Se almeno uno di questi valori è vero, la struttura IF visualizza una data; altrimenti la data non è compresa nel mese corrente ed il programma visualizza uno spazio. L'esempio finale illustra l'uso di una clausola ELSEIF in un selettore un po' più complesso. L'esempio è preso da una funzione che si chiama *LeggiLiCom*, che legge la linea di comando che l'utente digita dal DOS. Una delle opzioni dell'utente è fornire un singolo parametro intero. In questo caso, il programma legge l'intero come una richiesta per la visualizzazione di un solo mese o di un anno intero, a seconda del valore dell'intero. Se l'intero *inVal%* è all'interno dei mesi dell'intervallo (da 1 a 12), il valore viene registrato come una richiesta del mese. Altrimenti, se *inVal%* è all'interno dell'intervallo degli anni che il programma può gestire, la clausola ELSEIF registra il valore come un anno. Comunque, se né la condizione IF né ELSEIF sono valutate come vere, la clausola ELSE viene eseguita, registrando l'input dell'utente come non valido:

```
IF inVal% >= 1 AND inVal% <= 12 THEN
   qualeOutput% = meseAtt%
   mese% = inVal%
   anno% = 0
```

```
ELSEIF inVal% >= 1900 AND inVal% <= 2500 THEN
   qualeOutput% = annoCompl%
   anno% = inVal%
   mese% = 0
ELSE
   qualeOutput% = comErr%
   anno% = 0
   mese% = 0
END IF
```

La variabile *qualeOutput%* in questo frammento è un codice intero (da 0 a 5) che l'utente del programma usa per registrare il tipo di emissione che ha richiesto.

Alcuni esempi dell'istruzione IF su una sola linea

Il programma *Mese* contiene anche alcuni esempi interessanti di selettori che occupano una sola linea. La seguente istruzione è presa da una funzione chiamata *NumData&* che ritorna un intero che rappresenta una data assegnata. L'istruzione determina se aggiungere un giorno in più per un anno bisestile:

```
IF AnnoBisest%(annoCorr%) THEN num& = num& + 1
```

Si noti che la condizione in questa istruzione è una chiamata alla funzione *AnnoBisest%*. Questa funzione ritorna un intero che il programma legge come un valore vero o falso. Un secondo esempio, della funzione *NumGiorno%* dà il giorno della settimana (rappresentato come un intero da 1 a 7) di una data indicata:

```
IF g& <> 0 THEN gds% = g& MOD 7 + 1 ELSE gds% = 0
```

Il valore di *g&* è un intero lungo che rappresenta una data; questo valore è ottenuto da una chiamata alla funzione *NumData&*. *NumData&* ritorna zero per una data non valida; perciò la clausola ELSE in questo selettore ritorna zero per il giorno della settimana. Comunque, se *g&* non è zero, si suppone che la data sia valida e la funzione *NumGiorno%* ritorna un valore compreso tra 1 e 7.

Compatibilità

Il Turbo Basic offre i selettori IF sia strutturati a blocchi sia su una sola linea, come i selettori IF in QuickBASIC. BASICA, d'altra parte, supporta solo l'istruzione IF che occupa una sola linea. Le clausole THEN e ELSE delle istruzioni IF nei programmi BASICA spesso contengono istruzioni GOTO, perché il linguaggio interpretato BASICA non ha le istruzioni del selettore strutturato a blocchi disponibili in QuickBASIC.

SELECT CASE

Il selettore SELECT CASE si utilizza per organizzare una serie di opzioni mutuamente esclusive in blocchi distinti separati da CASE. Durante l'esecuzione di un programma, viene selezionata solo una delle opzioni CASE (se ce ne sono) ed eseguito il corrispondente frammento di codice.

```
SELECT CASE espressione
   CASE listaEspr
      ' primo blocco CASE
   CASE listaEspr
      ' secondo blocco CASE
   ' ... blocchi CASE aggiuntivi
   CASE ELSE     ' La clausola CASE ELSE è opzionale.
      ' blocco CASE ELSE

END SELECT
```

All'inizio della struttura SELECT CASE, nella stessa clausola SELECT CASE, si fornisce il valore che verrà usato per selezionare uno dei blocchi CASE seguenti. Si può esprimere questo valore SELECT CASE come una costante (numerica o stringa), come una variabile semplice o come un'espressione più complessa. All'interno della struttura SELECT CASE può esserci un numero qualsiasi di blocchi CASE. All'inizio di ogni blocco, una clausola CASE contiene una lista di uno o più valori, tutti appartenenti allo stesso tipo di dati del valore usato nella clausola SELECT CASE. Così come il valore SELECT CASE, i valori nelle clausole CASE possono essere costanti, variabili o espressioni. Inoltre, per esprimere intervalli e condizioni in una clausola CASE si possono usare le parole chiave TO e IS. La seguente notazione indica un *intervallo* di valori:

```
val1 TO val2
```

In questa espressione, *val1* è minore di *val2*. Se *val1* e *val2* sono stringhe, *val1* deve avere un valore ASCII inferiore a quello di *val2*. La parola chiave IS, usata con uno degli operatori relazionali di QuickBASIC, esprime una condizione in una clausola CASE:

```
IS op espressione
```

In questa notazione, *op* è uno dei sei operatori relazionali: <, >, <=, >=, <> o =. Le espressioni TO e IS possono presentarsi da sole oppure possono essere unite ad altre espressioni nella clausola CASE. Un blocco CASE è costituito da una sequenza di istruzioni che si trovano tra una clausola CASE e la successiva o tra la clausola finale CASE e END SELECT. In una clausola CASE si può usare qualsiasi istruzione eseguibile di QuickBASIC, comprese le strutture nidificate come i cicli, i selettori e anche le SELECT CASE. Durante un'esecuzione viene selezionato un particolare blocco CASE quando il valore dopo SELECT CASE corrisponde ad uno dei valori in una lista CASE. Dopo aver scelto ed eseguito un blocco CASE, tutti i rimanenti blocchi CASE vengono ignorati; il controllo del programma salta alla prima istruzione dopo la clausola END SELECT. Si può includere anche un blocco opzionale CASE ELSE alla fine della struttura SELECT CASE. QuickBASIC esegue questo blocco solo nel caso in cui nessuno dei blocchi CASE precedenti venga selezionato per l'esecuzione, cioè il valore SELECT CASE non corrisponda ad alcun valore di nessuna clausola CASE. Diversamente, se non si include un blocco CASE ELSE, nel caso in cui non venga trovata nessuna corrispondenza, la struttura SELECT CASE non risulterà in nessuna operazione.

Un esempio di SELECT...CASE

La seguente struttura **SELECT CASE** è un frammento del programma *Mese*, presentato nel Capitolo 29. Precisamente, è presa dal sottoprogramma *Output*, che controlla l'output del programma:

```
  SELECT CASE operaz%
```

```
' Visualizza il mese corrente ed il successivo
' se l'utente non ha dato nessuna istruzione o se non
' sono valide.

  CASE IS <= nessunCom%
     meseCorr% = VAL(LEFT$(DATE$, 2))
     annoCorr% = VAL(RIGHT$(DATE$, 4))
     StampOVisual Output%, meseCorr%, annoCorr%
     IF meseCorr% = 12 THEN
        meseCorr% = 1
        annoCorr% = annoCorr% + 1
     ELSE
        meseCorr% = meseCorr% + 1
     END IF
     StampOVisual Output%, meseCorr%, annoCorr%

' Visualizza un solo mese.

  CASE unMese%, questoMese%
     IF an% = THEN anno% = VAL(RIGHT$(DATE$, 4)) ELSE anno% = an%
     StampOVisual Output%, me%, anno%

' Visualizza tutti i dodici mesi dell'anno specificato.

  CASE annoCompl%
     FOR mese% = 1 TO 12
        StampOVisual Output%, mese%, an%
     NEXT mese%

END SELECT
```

Il compito di questa particolare struttura è scegliere una delle opzioni disponibili per l'output del programma. La scelta si basa sul valore della variabile intera *operaz%*, un valore che rappresenta il formato delle istruzioni date dall'utente al programma sulla linea di comando. Il programma *Mese* dichiara molte costanti simboliche per rappresentare le varie opzioni di output che l'utente può selezionare. Ad esempio, *nessunCom%* significa che l'utente non ha immesso nessuna istruzione sulla linea di comando. Le costanti *unMese%* e *questoMese%* indicano che la richiesta riguarda tutti dodici i mesi di un particolare anno. Le clausole CASE in questo esempio usano queste costanti per esprimere possibili valori di comparazione per *operaz%*. Si noti la varietà di formati CASE presentata in questo esempio. Il seguente blocco CASE verrà selezionato se *operaz%* è un valore che è minore o uguale a *nessunCom%*:

```
CASE IS <= nessunCom%
```

Questo successivo blocco CASE viene selezionato se *operaz%* è uguale a *unMese%* o *questoMese%*:

```
CASE unMese%, questoMese%
```

Infine, l'ultimo blocco CASE è selezionato se *operaz%* è uguale a *annoCompl%*:

```
CASE annoCompl%
```

Poiché questo esempio non contiene nessuna clausola CASE ELSE, la struttura non risulta in nessuna operazione quando non si trova un uguaglianza tra la variabile *operaz%* ed uno dei valori CASE.

Compatibilità

La clausola SELECT CASE di Turbo Basic è quasi identica a quella di QuickBASIC. L'unica differenza è che, in Turbo Basic, un'espressione relazionale in una clausola CASE non usa la parola chiave IS. Ad esempio, si consideri questa clausola QuickBASIC:

```
CASE IS <= nessunCom%
```

In Turbo Basic la stessa clausola si presenterà semplicemente così:

```
CASE <= nessunCom%
```

Non c'è alcuna struttura SELECT CASE in BASICA. Comunque, l'istruzione ON...GOSUB ha un ruolo simile. (ON...GOSUB è disponibile anche in QuickBASIC.)

Capitolo 8

Altre istruzioni di controllo

Questo capitolo comprende le voci relative a molte tradizionali strutture di controllo BASICA, supportate in QuickBASIC per compatibilità con le versioni precedenti del linguaggio:

- Il salto GOTO e le chiamate al sottoprogramma GOSUB.
- Struttura delle Funzioni definite dall'utente DEF FN.
- Selettori ON...GOTO e ON...GOSUB.

Per lo più, queste istruzioni sono ormai sorpassate nell'ambiente di programmazione strutturata di QuickBASIC 4.5. La maggior parte dei programmatori evita di usarle, tranne che nei programmi che richiedono compatibilità con l'interprete BASICA o in programmi che usano tecniche di gestione degli errori e degli eventi. (Si possono leggere notizie riguardanti la gestione di errori ed eventi nella Parte VI di questo libro.) Due voci aggiuntive in questo capitolo descrivono degli elementi anomali, ma importanti del linguaggio di QuickBASIC. I comandi EXIT, qui descritti tutti insieme in un'unica voce, hanno il compito di bloccare l'esecuzione di una procedura QuickBASIC o di un ciclo. Questi comandi vengono riservati per situazioni speciali; ma occasionalmente possono essere utili per semplificare l'organizzazione interna di una procedura o di un ciclo. Infine, la funzione COMMAND$ (discussa nel primo paragrafo di questo capitolo) fornisce al programma compilato le istruzioni che l'utente immette sulla

linea di comando DOS. In un certo senso, la funzione COMMAND$ ricopre un ruolo simile a quello della lista dei parametri nella definizione di una procedura: entrambe hanno il compito di precisare le modalità di comunicazione di dati tra diversi livelli di controllo di un programma.

COMMAND$

La funzione COMMAND$ ritorna un valore stringa costituito dal testo che l'utente immette sulla linea di comando del DOS dopo il nome del programma. La funzione non ha argomenti:

```
COMMAND$
```

Si può usare COMMAND$ per far sì che un programma reagisca a specifiche istruzioni dalla linea di comando del prompt del DOS. La stringa COMMAND$ fornisce parola per parola il testo che l'utente introduce immediatamente dopo il nome del programma, con due eccezioni:

- QuickBASIC elimina gli spazi dall'inizio della stringa. (Gli spazi terminali vengono lasciati intatti.)
- Ogni lettera minuscola nella stringa della linea di comando viene automaticamente trasformata in maiuscola.

Ad esempio, si immagini un programma, di nome PROGRAM.BAS, con il compito di eseguire alcune operazioni su un valore stringa fornito dal prompt del DOS. La versione compilata del programma verrebbe memorizzata su disco sotto il nome di PROGRAM.EXE. Un utente potrebbe usare il seguente comando del prompt del DOS per inizializzare un'esecuzione di questo programma:

```
C>PROGRAM xyz
```

In questo caso, una chiamata alla funzione COMMAND$ fornirebbe il valore stringa maiuscolo "XYZ". Mentre si sta sviluppando un programma, si può usare la selezione *Modify COMMAND$...* dal menù Run di QuickBASIC per assegnare dei valori alla funzione COMMAND$. Quando

si sceglie questo comando, sullo schermo appare una finestra di dialogo per l'input in cui si può introdurre il valore stringa per COMMAND$. (Questa finestra di dialogo accetta una stringa fino a 127 caratteri, la lunghezza massima di una linea di comando del DOS.) Quando successivamente si attiva il programma all'interno dell'ambiente di sviluppo di QuickBASIC, la funzione COMMAND$ ritorna questa stringa. Più tardi, quando il proprio lavoro di sviluppo è completo, si può compilare il programma come un file EXE che può essere eseguito direttamente dal DOS. A questo punto, COMMAND$ ritorna l'effettiva linea di comando del DOS.

Un esempio della funzione COMMAND$

Il programma *Mese*, presentato nel Capitolo 29, ha il compito di leggere le istruzioni dell'utente fornite come argomenti sulla linea di comando DOS. *Mese* visualizza un solo mese di un calendario passato o futuro, un paio di mesi dal calendario corrente o un intero anno, a seconda delle istruzioni che l'utente fornisce sulla linea di comando. Inoltre, il programma è in grado di visualizzare su video il suo output oppure inviarlo alla stampante, sempre secondo le istruzioni dell'utente. Il programma contiene il sottoprogramma *LeggiLinCom*, che legge la linea di comando dell'utente e la separa in singole voci. All'inizio della procedura, la seguente istruzione memorizza il valore COMMAND$ nella variabile stringa *linCom$:*

```
linCom$ = RTRIM$(COMMAND$)
```

Si noti che questa istruzione usa la funzione RTRIM$ per togliere ogni spazio terminale dalla linea di comando. Il resto della procedura *LeggiLinCom* ha il compito preciso di determinare quali informazioni sono effettivamente contenute nella linea di comando. Vedere il programma *Mese* per ulteriori informazioni.

Compatibilità

Anche Turbo BASIC fornisce la funzione COMMAND$, ma non esegue la trasformazione da maiuscole in minuscole o viceversa che invece avviene automaticamente in QuickBASIC. BASICA non ha la funzione COMMAND$.

DEF FN

La struttura DEF FN definisce una funzione in un formato che è compatibile con le versioni 3.0 e precedenti del QuickBASIC. Il compito di una funzione DEF FN è ritornare un solo valore di un tipo specificato. (QuickBASIC non tratta una funzione DEF FN come una procedura separata, come fa con le strutture FUNCTION...END FUNCTION. In generale, le strutture DEF FN sono meno potenti ed utili delle procedure FUNCTION.) La struttura DEF FN può comparire in un formato ad una sola linea o a più linee.

Formato a più linee:

```
DEF nomeFunzFN (parametri)   ' La lista dei parametri è opzionale.

   ' blocco di istruzioni QuickBASIC

   nomeFunzFN = espressione

END DEF
```

Formato ad una sola linea:

```
DEF nomeFunzFN (parametri) = espressione
```

Nel formato a più linee, le istruzioni DEF FN e END DEF segnano l'inizio e la fine della definizione della funzione. Tra queste due linee si può inserire un blocco di quante istruzioni QuickBASIC si desidera. Le variabili usate all'interno della funzione sono globali per default, cioè sono disponibili ovunque nel programma. (Per ulteriori informazioni vedere il paragrafo "Campo di visibilità" nel Capitolo 2.) L'istruzione DEF FN definisce il nome di una funzione e specifica anche il tipo di valore che la funzione ritornerà. Un nome può essere formato al massimo da 40 caratteri, inizia con una lettera ma può comprendere lettere e/o cifre. Il carattere finale indica il tipo di valore che la funzione ritornerà: & per un intero lungo, % per un intero, # per la doppia precisione, ! (o nessun tipo speciale di carattere) per la virgola mobile in precisione semplice e $ per un valore stringa. Si può fornire una lista opzionale di parametri, messa tra parentesi nell'istruzione DEF FN. Questi parametri rappresentano i valori dei dati che saranno passati alla funzione tramite una chiamata. Le voci nella lista dei parametri possono

essere semplici nomi di variabili con caratteri di tipo (&, %, #, ! o $). Oppure si può usare la seguente notazione per specificare il tipo di un parametro:

```
nomeVariab AS parchiaveTipo
```

dove le parole chiave tipo disponibili sono LONG, INTEGER, DOUBLE, SINGLE e STRING. Diversamente da una procedura FUNCTION, una funzione DEF FN non può ricevere vettori o record come argomenti. Per specificare il valore di ritorno, un'istruzione all'interno della funzione assegna un valore al nome della funzione:

```
nomeFunzFN = espressione
```

Questa istruzione appare alla fine del blocco di funzione, ma effettivamente può essere messa in qualsiasi punto all'interno della funzione. Nel formato ad una linea di DEF FN, questa espressione è l'elemento finale della stessa istruzione DEF FN:

```
DEF nomeFunzFN (parametri) = espressione
```

Una chiamata della funzione DEF FN è rappresentata dal postfisso FN, preceduto dal nome della funzione e da una lista appropriata di argomenti fra parentesi:

```
nomeFunzFN(argomenti)
```

Una chiamata di funzione è sempre parte di un'espressione o di un'istruzione che usa il valore ritornato dalla funzione. Il programma deve definire la funzione DEF FN prima di provare una chiamata di funzione. Si può esprimere un valore dell'argomento come una costante, una variabile o un'espressione. Indipendentemente dal formato, gli argomenti sono sempre passati ad una funzione DEF FN *per valore*. Ciò significa che QuickBASIC invia una copia del valore dell'argomento e la funzione DEF FN non può far tornare indietro, al resto del programma, i cambiamenti nel valore. Nelle funzioni DEF FN le chiamate ricorsive non sono ammesse. Riassumendo, ecco gli svantaggi della struttura DEF FN:

1. QuickBASIC non tratta una funzione DEF FN come una procedura, e perciò non fornisce una finestra di visualizzazione separata per la funzione.

2. Una funzione DEF FN non può accettare vettori o record come argomenti.
3. Tutti gli argomenti vengono passati per valore ad una funzione DEF FN; il passaggio per indirizzo non è disponibile.
4. Le variabili usate all'interno di una funzione DEF FN sono globali per default.
5. Le chiamate ricorsive DEF FN non sono ammesse.

La procedura FUNCTION...END FUNCTION è più adatta allo stile strutturato di programmazione generalmente usato in QuickBASIC 4.5. È meglio evitare di usare le definizioni DEF FN se non necessarie, per compatibilità con un'altra versione del linguaggio.

Un esempio di una funzione DEF FN

La seguente funzione multilinee DEF FN è equivalente alla prima funzione esempio presentata nel paragrafo FUNCTION. Questa funzione, chiamata *AnnoBisestFN%*, riceve un solo argomento intero che rappresenta un anno e ritorna un valore logico che indica se l'argomento è un anno bisestile o meno:

```
DEF AnnoBisestFN% (anno%)

' La funzione AnnoBisest% ritorna un valore booleano che
' indica se un determinato anno è bisestile.

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)

   AnnoBisestFN% = divPer4% AND (secolo% IMP secolo400%)

END DEF
```

Il seguente frammento di programma esegue molte chiamate alla funzione *AnnoBisestFN%* e visualizza tutti gli anni bisestili dal 1980 al 2000:

```
CLS
PRINT "Anno Bisestile: ";
```

```
FOR i = 1980 TO 2000
   IF AnnoBisestFN%(i) THEN PRINT i;
NEXT i
```

Ecco l'output da questo ciclo:

```
Anni Bisestili: 1980 1984 1988 1992 1996 2000
```

Vedere la voce FUNCTION...END FUNCTION per ulteriori informazioni su questa routine.

Compatibilità

Turbo Basic permette sia le funzioni ad una linea sia le multilinee; ma non supporta la struttura FUNCTION...END FUNCTION. BASICA consente solo le funzioni DEF FN ad una linea.

EXIT

Le varie forme dell'istruzione EXIT sono disponibili per bloccare una procedura ciclo QuickBASIC.

Uscite di una procedura:

```
EXIT SUB          ' Uscita da un sottoprogramma.
EXIT FUNCTION     ' Uscita da una funzione.
EXIT DEF          ' Uscita da una struttura DEF FN.
```

Uscite di un ciclo:

```
EXIT FOR          ' Uscita da un ciclo FOR.
EXIT DO           ' Uscita da un ciclo DO.
```

Nei sottoprogrammi, nelle funzioni e nelle strutture DEF FN, le istruzioni EXIT rimandano il controllo alla posizione della chiamata originale. Ad esempio, EXIT SUB fa sì che QuickBASIC riprenda l'esecuzione dall'istruzione che si trova immediatamente dopo la chiamata al sottoprogramma. Nei cicli, EXIT invia il controllo all'istruzione posta subito dopo il ciclo.

Specificatamente, EXIT FOR invia il controllo alla linea dopo l'istruzione NEXT; e EXIT DO salta alla linea dopo l'istruzione LOOP. Le istruzioni EXIT vanno usate con parsimonia per un buon stile di programmazione. Nella maggior parte dei casi la struttura del programma sarà più chiara e più leggibile se si utilizzano selettori nidificati al posto di EXIT per controllare la logica di una procedura o di un ciclo. Di quando in quando, comunque, un'istruzione EXIT può essere usata per semplificare l'organizzazione di un particolare passaggio del programma.

Un esempio di un'istruzione EXIT

Un uso appropriato di un'istruzione EXIT è determinare il funzionamento di un sottoprogramma o di una funzione nel caso in cui la procedura ha ricevuto valori dell'argomento non validi. Questa tecnica è usata nel programma *Mese* (presentata nel Capitolo 29). Ad esempio, si consideri la funzione *NumData&*, il cui compito è calcolare un intero per rappresentare una data. Se gli argomenti ricevuti dalla funzione non rappresentano una data valida, la seguente istruzione IF termina l'esecuzione della funzione:

```
IF tropPres% OR meseErr% OR giornErr% THEN
   NumData& = 0
   EXIT FUNCTION
END IF
```

Proprio prima di questa struttura del selettore, il programma assegna valori logici alle tre variabili *tropPres%*, *meseErr%* e *giornErr%*. Se uno di questi valori è vero (cioè, se la data è precedente alla prima data che la routine può gestire; se l'intero del mese non è compreso nell'intervallo 1-12; o se l'intero del giorno non è incluso nell'appropriato intervallo dei giorni del mese specificato), la funzione *NumData&* ritorna un valore pari a zero, e l'istruzione EXIT FUNCTION termina l'operazione della procedura. Comunque, se tutte queste condizioni sono false, la funzione continua a calcolare la rappresentazione dell'intero della data specificata.

Compatibilità

Turbo Basic supporta tutte le stesse istruzioni EXIT ad eccezione di EXIT DEF. Inoltre, Turbo Basic include le seguenti istruzioni EXIT per selettori:

```
EXIT IF       ' Uscita da un blocco IF.
EXIT SELECT   ' Uscita da una struttura SELECT CASE.
```

BASICA non supporta EXIT.

GOSUB E RETURN

L'istruzione GOSUB invia il controllo all'inizio di un sottoprogramma indicato da un numero di linea o da un'etichetta. Una chiamata sottoprogramma termina con un'eventuale istruzione RETURN, che ritorna il controllo ad una locazione specificata.

Chiamata ad un sottoprogramma:

```
GOSUB numLin       ' Per un sottoprogramma che inizia da
                   ' un numero di linea.
```

o

```
GOSUB linEtich     ' Per un sottoprogramma che inizia da
                   ' un'etichetta.
```

Restituzione del controllo da un sottoprogramma

```
RETURN             ' Ritornare all'istruzione dopo
                   ' GOSUB.
```

o

```
RETURN numLin      ' Ritornare ad un numero di linea
                   ' specificato.
```

o

```
RETURN numEtich   ' Ritornare alla locazione di un'etichetta.
```

In un programma che contiene numeri di linea, GOSUB invia il controllo ad una linea di numero specificato. In alternativa, GOSUB può inviare il controllo alla locazione di un'etichetta specificata. Si può includere un'etichetta in ogni punto del listato del programma, nel seguente formato:

```
nomeEtich
```

Come il nome di una variabile, l'etichetta di una linea può contenere fino a 40 lettere e/o cifre, ma deve iniziare con una lettera. I due punti richiesti alla fine identificano il nome come un'etichetta. L'istruzione RETURN appare all'interno del flusso di controllo dopo l'etichetta o il numero di linea che rappresenta l'inizio del sottoprogramma. Per default, RETURN invia il controllo all'istruzione immediatamente seguente l'originale GOSUB. In alternativa, si può includere un numero di linea opzionale o un'etichetta nell'istruzione RETURN per specificare una diversa locazione di ritorno. I GOSUB possono anche essere localizzati all'interno di un sottoprogramma strutturato (SUB...END SUB) o di una funzione (FUNCTION...END FUNCTION). Comunque, GOSUB non può inviare il controllo ad una locazione che si trova in un altro sottoprogramma o funzione, cioè, le chiamate GOSUB non possono passare da una procedura QuickBASIC all'altra. Inoltre, la sintassi alternativa di RETURN, che specifica il numero di linea o l'etichetta, non è ammessa all'interno di un sottoprogramma o di una funzione, solo a livello del programma principale.

Un esempio di GOSUB e RETURN

Il seguente esercizio è una revisione del sottoprogramma *Output*, una routine che controlla l'emissione dal programma *Mese* (listato nel Capitolo 29). Dopo aver determinato se l'emissione deve essere eseguita su schermo o stampante, il sottoprogramma originale *Output* invia chiamate a due altri sottoprogrammi, *PrepStamp* e *StampOVisual*, per completare il processo di emissione. In questa nuova versione del sottoprogramma, queste due procedure sono state riscritte e chiamate attraverso le istruzioni GOSUB:

```
SUB Output

' La procedura Output seleziona una delle molte scelte di output,
' secondo le istruzioni della linea di comando indicate dall'utente.
```

```
LeggiLiCom operaz%, me%, an%

IF operaz% >= stampaInf% THEN
   Output% = vero%
   operaz% = operaz% - stampaInf%
   GOSUB PrepStamp
ELSE
   Output% = falso%
END IF
PRINT

ON operaz% + 1 GOTO ComandErr, nessunCom, UnMese, meseAtt, AnnoInt

' Visualizza il mese corrente ed il successivo se l'utente non ha
' dato nessuna istruzione o se le istruzioni dell'utente non
' sono valide.

ComandErr:
NessunCom:

   meseCorr% = VAL(LEFT$(DATE$, 2))
   annoCorr% = VAL(RIGHT$(DATE$, 4))
   GOSUB StampOVisual
   IF meseCorr% = 12 THEN
      meseCorr% = 1
      annoCorr% = annoCorr% + 1
   ELSE
      meseCorr% = meseCorr% + 1
   END IF
   GOSUB StampOVisual
   GOTO FineSuGoto

UnMese:
MeseAtt:

   IF anno% = 0 THEN annoCorr% = VAL(RIGHT$(DATE$, 4))
               ELSE annoCorr% = anno%
   GOSUB StampOVisual
   GOTO FineSuGoto

AnnoInt:

   annoCorr% = VAL(RIGHT$(DATE$, 4))
   FOR mese% = 1 TO 12
      meseCorr% = mese%
      GOSUB StampOVisual
   NEXT mese%
   GOTO FineSuGoto

PrepStamp:
```

```
' Il sottoprogramma PrepStamp dà all'utente la possibilità
' di preparare la stampante.

   PRINT "Premere la barra spaziatrice quando la stampante è pronta. ";
   barSp$ = ""
   DO WHILE barSp$ <> " "
      barSp$ = INKEY$
   LOOP
   PRINT
   RETURN

StampOVisual:

' Il sottoprogramma StampOVisual chiama o la routine StampMese
' per stampare un mese, o la routine VisualMese
' per visualizzare un mese sullo schermo.

   IF Output% THEN
      StampMese meseCorr%, annoCorr%
      LPRINT
   ELSE
      VisualMese meseCorr%, annoCorr%
      PRINT
   END IF
   RETURN

FineSuGoto:

END SUB
```

Le variabili all'interno di un sottoprogramma sono condivise con la procedura in cui si trova il sottoprogramma stesso. Ad esempio, il sottoprogramma *Output* comunica con il sottoprogramma *StampOVisual* attraverso tre variabili condivise, *Output%*, *meseCorr%* e *annoCorr%*. Poiché un'istruzione GOSUB non può passare argomenti, una variabile condivisa è l'unico tipo di strumento possibile per rendere disponibili i dati al sottoprogramma chiamato. Si noti anche che la procedura *Output* richiede una sequenza di comandi GOTO *(FineSuGoto GOTO)* per evitare un'esecuzione non voluta di un sottoprogramma. Questo esempio, con le complicazioni di GOTO e la confusione di variabili condivise, illustra chiaramente perché le procedure SUB...END SUB sono superiori ai sottoprogrammi.

Compatibilità

Sia Turbo Basic sia BASICA supportano i sottoprogrammi. I programmi BASICA hanno sempre i numeri di linea; le etichette di linea non vengono usate.

GOTO

GOTO invia il controllo ad un numero o ad un'etichetta di linea. Questo comando BASICA dovrebbe essere evitato in programmi strutturati QuickBASIC:

```
GOTO numLinea
```

o

```
GOTO etichLinea
```

In un programma che contiene numeri di linea, GOTO invia il controllo ad una particolare linea numerata. In alternativa, GOTO può saltare alla posizione di un'etichetta. Si può includere un'etichetta in ogni punto del listato di un programma nel seguente formato:

```
nomeEtich:
```

Come il nome di una variabile, un'etichetta di linea può contenere fino a 40 lettere e/o cifre, ma deve iniziare con una lettera. I due punti richiesti alla fine identificano il nome come un'etichetta. Le istruzioni GOTO possono essere messe all'interno della struttura di un sottoprogramma (SUB...END SUB) o di una procedura di funzione (FUNCTION...END FUNCTION). Comunque, GOTO non può saltare da una procedura all'altra.

Un esempio di GOTO

L'esempio di programma presentato nel paragrafo GOSUB contiene molti esempi di istruzioni GOTO usate per evitare esecuzioni non volute di sottoprogrammi. Ecco un frammento del programma:

```
   GOTO FineSuGoto
PrepStamp:
'    ...
   RETURN
StampOVisual:
'    ...
   RETURN
FineSuGoto:
END SUB
```

Senza l'istruzione *GOTO FineSuGoto*, il flusso del programma passerà inavvertitamente al primo sottoprogramma. Comunque, una soluzione decisamente migliore è illustrata nel programma *Mese* (presentato nel Capitolo 29), che evita l'uso di GOTO, GOSUB e ON...GOBUB tutti insieme.

Compatibilità

GOTO è disponibile, ma può e dovrebbe essere evitato, in Turbo Basic. Nel linguaggio interpretato BASICA, GOTO è una parte essenziale della maggior parte dei programmi.

ON...GOSUB E ON...GOTO

ON...GOSUB e ON...GOTO sono selettori che saltano in un punto selezionato del programma, a seconda del valore di un'espressione ON. (Queste strutture sono disponibili per compatibilità con QuickBASIC 2.0 o precedenti. In QuickBASIC 4.5 si dovrebbe usare invece la struttura SELECT CASE per implementare un selettore di questo tipo.)

```
ON espress GOSUB listLinee

ON espress GOTO listLinee
```

In ognuna di queste istruzioni, l'*espressione* è una variabile o un'espressione che assume il valore di un intero positivo. (Se il valore non è un intero, QuickBASIC lo arrotonda all'intero più prossimo.) Una lista di numeri e/o etichette di linea appare immediatamente dopo la parola chiave GOTO o GOSUB. QuickBASIC usa la variabile che segue ON per scegliere uno dei numeri o delle etichette di linea in questa lista: se il valore è 1, l'istruzione salta alla prima linea della lista; se è 2 alla seconda linea e così via. Nella lista ci possono essere fino a 60 numeri e/o etichette di linea. Se l'espressione ON assume il valore di zero o un valore che è più grande del numero di linee indicate nella lista, non si ha nessun salto. Si verifica un errore run-time se l'espressione ON è un valore minore di zero o maggiore di 255. ON...GOTO salta in una nuova posizione all'interno del programma e non assume nessun impegno a proposito di un eventuale ritorno del controllo. Al contrario, ON...GOSUB esegue una chiamata di subroutine: un'istruzione RETURN dovrebbe eventualmente essere inclusa per ritornare ad una specifica posizione. In caso contrario, la linea di programma a cui viene inviato il controllo può essere identificata con un numero di linea o con una etichetta nella forma:

```
nomeEtich:
```

Per ulteriori informazioni su queste operazioni, vedere le voci GOTO e GOSUB.

Un esempio dell'istruzione ON...GOTO

Il seguente frammento è una revisione di un esempio presentato nel paragrafo SELECT CASE:

```
   ON operaz% + 1 GOTO ComandErr, nessunCom, UnMese, meseAtt, AnnoInt

ComandErr:
nessunCom:

   meseCorr% = VAL(LEFT$(DATE$, 2))
   annoCorr% = VAL(RIGHT$(DATE$, 4))
   StampOVisual Output%, meseCorr%, annoCorr%
   IF meseCorr% = 12 THEN
      meseCorr% = 1
```

```
      annoCorr% = annoCorr% + 1
   ELSE
      meseCorr% = meseCorr% + 1
   END IF
   StampOVisual Output%, meseCorr%, annoCorr%
   GOTO FineSuGoto

UnMese:
MeseAtt:

   IF anno% = 0 THEN annoCorr% = VAL(RIGHT$(DATE$, 4)) ELSE annoCorr% = anno%
   StampOVisual Output%, mese%, anno%
   GOTO FineSuGoto

AnnoInt:

   FOR mese% = 1 TO 12
      meseCorr% = mese%
      StampOVisual Output%, mese%, anno%
   NEXT mese%
   GOTO

FineSuGoto:
```

Nell'istruzione ON...GOTO all'inizio di questo frammento, l'espressione *operaz% + 1* assume il valore di un intero da 1 a 5. Di risposta, QuickBASIC sceglie una delle cinque etichette di linea nella lista GOTO. Due coppie di etichette (*ComandErr* e *NessunCom*; *UnMese* e *MeseAtt*) rappresentano effettivamente gli stessi blocchi di codice. Si noti che serve un'ulteriore istruzione GOTO alla fine di ogni blocco per inviare il controllo alla fine della routine (*FineSuGoto*) dopo che un dato blocco è stato eseguito. Si confronti questa tecnica piuttosto rozza con quella più chiara dell'istruzione SELECT CASE.

Compatibilità

Turbo Basic supporta sia le istruzioni ON...GOTO/ON...GOSUB sia SELECT CASE. BASICA offre solo ON...GOTO e ON...GOSUB.

Parte III

Input e output

Le tecniche per l'input e l'output sono tra le più importanti. In generale, per input si intendono le operazioni con cui un programma acquisisce dati, da ogni tipo di fonte; viceversa, l'output è il mezzo con cui un programma indirizza le informazioni all'esterno verso una particolare destinazione. I sei capitoli della Parte III esaminano le fonti e le destinazioni più comuni dei dati, le istruzioni e le funzioni QuickBASIC che eseguono compiti di input e output:

Il Capitolo 9 prende in esame l'insieme degli strumenti che QuickBASIC fornisce per leggere l'input da tastiera.

Il Capitolo 10 tratta delle istruzioni e delle funzioni che visualizzano le informazioni sullo schermo o inviano i dati alla stampante.

Il Capitolo 11 considera l'argomento della gestione dei file di dati in QuickBASIC. Specificatamente, le voci di questo capitolo riguardano tutte le istruzioni e funzioni che si possono usare per gestire file ad accesso sequenziale e diretto.

Il Capitolo 12 riconsidera alcune vecchie tecniche stile BASICA per la gestione di file di dati. Le istruzioni e le funzioni qui discusse sono ancora disponibili, ma ormai superate dall'introduzione di strutture record di QuickBASIC.

Il Capitolo 13 discute le tecniche di input e output per dispositivi hardware speciali.

Il Capitolo 14 presenta gli strumenti QuickBASIC che si possono usare per produrre suoni e musica dall'altoparlante del proprio computer.

Nella Parte VIII si troverà un programma completo interattivo chiamato *Agenda* (Capitolo 30), che illustra molte delle tecniche input e output qui discusse. Questo programma ha il compito di aiutare l'utente a tenere un calendario per gli appuntamenti di lavoro, per la casa o per le attività d'ufficio. Il programma memorizza in un archivio su disco le registrazioni relative ai giorni del calendario e consente all'utente di esaminare, creare e riesaminare gli appuntamenti per ogni data del presente o del futuro. Un programma associato chiamato *GestioneDate* (Capitolo 31) costituisce un'utile interfaccia con l'utente, per il programma *Agenda*. Infine, il quinto principale esempio di programma, chiamato *DateStoriche* (Capitolo 32), contiene alcuni esempi interessanti di istruzioni QuickBASIC che producono suoni, come descritto nel Capitolo 14.

Capitolo 9

Input da tastiera

Questo capitolo esamina le istruzioni di QuickBASIC che leggono l'input da tastiera. Come gruppo, queste istruzioni sono tra le più importanti e le più comunemente usate nel linguaggio. Un programma *interattivo* accetta dati dalla persona che c'è alla tastiera e poi risponde appropriatamente all'input dell'utente. L'interazione tra programma e utente, talvolta, è chiamata *dialogo*. Per istaurare un dialogo, il programma deve essere in grado di visualizzare su schermo delle richieste, creare una pausa fino a quando l'utente non risponde da tastiera, leggere il valore o i valori immessi e tenere in memoria le risposte dell'utente. QuickBASIC offre diverse istruzioni per eseguire questi compiti. A seconda delle necessità di un particolare dialogo d'input, un programma può leggere una singola battuta di tasto, una sequenza di valori immessi o un'intera linea di testo alla volta. In ogni caso, il funzionamento del programma si blocca finché l'utente non ha avuto la possibilità di introdurre da tastiera delle informazioni; poi la risposta viene memorizzata in una o più variabili. Le voci di questo capitolo descrivono le istruzioni e le tecniche per eseguire questi input in QuickBASIC.

INKEY$

La funzione INKEY$ legge la più recente immissione da tastiera. Se è stato premuto un tasto, la funzione ritorna una stringa a uno o due caratteri; in caso contrario, INKEY$ ritorna una stringa vuota.

INKEY$ non richiede alcun argomento. Il valore di ritorno dipende dal fatto che sia stato premuto un tasto della tastiera. I valori immessi da tastiera sono registrati in un'area di memoria conosciuta come il buffer della tastiera; INKEY$ legge da questo buffer. Se il buffer non contiene caratteri, INKEY$ ritorna una stringa vuota. Per questo motivo la funzione di solito appare come parte di un ciclo che continua a rileggere il buffer finché il valore di ritorno di INKEY$ non è più vuoto. INKEY$ non visualizza su video il carattere immesso: se si vuole che esso appaia sullo schermo, il programma deve usare un'istruzione PRINT per visualizzarlo. In risposta ad un tasto premuto che corrisponde ad un carattere ASCII, INKEY$ ritorna una stringa di un carattere. Un programma può usare la funzione ASC per determinare il numero di codice ASCII del tasto. (Nell'Appendice A si trova la tabella dei codici ASCII.) Per particolari tasti che non hanno un codice ASCII corrispondente, INKEY$ ritorna una stringa di due caratteri. Il primo carattere segnala che il tasto non è ASCII; il valore di questo flag è sempre il carattere nullo ASCII, che ha un codice 0. Il secondo carattere è un valore preso dal *codice esteso della tastiera*. Questo codice consente ad un programma di identificare speciali tasti come quelli di funzione, quelli di direzione del cursore e combinazioni di più tasti che comprendono Ctrl e Alt. Quando INKEY$ ritorna una stringa a due caratteri, la funzione ASC può essere usata per eseguire un test sui valori di entrambi i caratteri: ASC ritorna zero per il primo carattere e un valore del codice esteso della tastiera per il secondo carattere. Va fatta una precisazione per quanto riguarda la terminologia: nel contesto della programmazione BASIC una *stringa vuota*, con lunghezza pari a zero, talvolta è chiamata *stringa nulla*. Ciò, però, può ingannare. Una stringa vuota non è uguale ad una stringa che contiene il carattere ASCII noto come nullo, il cui codice è 0. Una stringa vuota è quella che INKEY$ ritorna se il buffer è vuoto. Al contrario, un carattere nullo è il primo dei due caratteri che INKEY$ fornisce di risposta ad un tasto che non corrisponde ad alcun codice ASCII.

Esempi di INKEY$

INKEY$ è uno strumento efficiente per leggere una risposta formata da un carattere digitato dall'utente alla tastiera. In particolare, un programma che

richiede all'utente di premere un tasto appartenente ad un certo insieme può chiamare la funzione INKEY$ più volte all'interno di un ciclo finché si riceve una battuta di tasto valida. Non risulta nessun output su schermo fino a quando il programma non visualizza esplicitamente il valore con un'istruzione PRINT. Il seguente breve frammento di programma è preso da *Agenda* (presentato nel Capitolo 30). Il programma usa un'istruzione PRINT per visualizzare su video una richiesta e richiedere una risposta affermativa o negativa. Poi il programma ripete le chiamate a INKEY$ fin che l'utente preme il tasto S o il tasto N.

```
PRINT "Salvo questi appuntamenti del giorno? <S> o <N>";
DO
   ris$ = UCASE$(INKEY$)
LOOP UNTIL LEN(ris$) > 0 AND INSTR("SN", ris$) <> 0
PRINT ris$;
```

Si noti l'uso della funzione UCASE$ per convertire il valore di ritorno di INKEY$ in lettere maiuscole:

```
ris$ = UCASE$(INKEY$)
```

Questa tecnica semplifica al programma la valutazione del valore del tasto premuto. Nella clausola UNTIL del ciclo DO vengono richieste due verifiche. Primo, il programma deve assicurarsi che un tasto sia stato effettivamente premuto, in altre parole, che la lunghezza di *ris$* sia maggiore di zero:

```
LEN(ris$) > 0
```

Poi il programma utilizza l'istruzione INSTR per scoprire se il tasto premuto è uno delle due risposte valide:

```
INSTR("SN", ris$) <> 0
```

(Per ulteriori informazioni sulle funzioni di queste due stringhe vedere le voci LEN e INSTR nella Parte V.) Infine, questo frammento finisce visualizzando su schermo la risposta valida dell'utente:

```
PRINT ris$
```

Questa schermata dà all'utente la conferma visiva della risposta selezionata della tastiera. Questo programma è un esercizio INKEY$ che aiuterà a capire

l'uso del codice esteso della tastiera. Il programma visualizza su video i tasti premuti ed indica se l'input è un codice ASCII o esteso:

```
' Programma test per la funzione INKEY$.

CONST ESC = 27, F1 = 59, DEL = 83
DIM scan$(F1 TO DEL)

FOR i% = LBOUND(scan$) TO UBOUND(scan$)
   READ scan$(i%)
NEXT i%

DO
   battuta$ = INKEY$
   SELECT CASE(battuta$)
      CASE 0
         LOCATE CSRLIN, 1
         PRINT "Attendere... (premere <Esc> per uscire)";
   CASE 1
      codice% = ASC(battuta$)
      CLS
      PRINT battuta$; " = "; codice%; "(ASCII)"
      PRINT
   CASE 2
      CLS
      codice% = ASC(MID$(battuta$, 2))
      IF codice% >= LBOUND(scan$) AND codice% <= UBOUND(scan$) THEN
         PRINT scan$(codice%); " = ";
      END IF
      PRINT codice% " (codice della tastiera)"
      PRINT
   END SELECT
LOOP UNTIL codice% = ESC

DATA F1, F2, F3, F4, F5, F6, F7, F8, F9, F10
DATA , , Home, Up, PgDn, Sx, , Dx
DATA , Fine, Down, PgUp, Ins, Canc
```

Quando si avvia il programma sullo schermo compare questo messaggio:

```
Attendere... (premere <Esc> per uscire)
```

Per esaminare il codice corrispondente si può premere qualsiasi tasto della tastiera. Ad esempio, se si preme il tasto della virgola (un carattere ASCII), compare questo messaggio:

```
, = 44 (ASCII)
```

La virgola ha come valore del codice ASCII 44. D'altra parte, se si preme il tasto con la freccia rivolta verso l'alto, compare invece questo messaggio:

```
Su = 72 (codice della tastiera)
```

Il codice della tastiera per il tasto della freccia rivolta verso l'alto è 72. In altre parole, quando si preme questo tasto, INKEY$ ritorna due caratteri, rispettivamente con valori ASCII pari a 0 e 72. Si può anche usare il programma per prendere in esame alcuni tasti di funzione e alcune combinazioni della tastiera in cui si usano Ctrl e Alt. Per terminare il programma bisogna premere il tasto Esc. In alcuni programmi può essere importante svuotare il buffer della tastiera prima di spostarsi in una nuova routine d'input. La seguente procedura, un frammento del programma *GestioneDate* presentato nel Capitolo 31, esegue il semplice, ma importante compito di cancellare tutti i caratteri dal buffer della tastiera:

```
SUB CancellaBuffer

' La procedura CancellaBuffer cancella tutti i caratteri dal buffer
' della tastiera.

  DO WHILE INKEY$ <> ""
  LOOP

END SUB
```

Questa procedura è formata da un ciclo DO vuoto. Ogni iterazione del ciclo chiama la funzione INKEY$ per leggere un nuovo carattere dal buffer d'input. Il ciclo finisce solo quando INKEY ritorna una stringa vuota, che indica che anche il buffer stesso è vuoto.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione INKEY$.

INPUT

L'istruzione INPUT visualizza sullo schermo una stringa opzionale del prompt ed aspetta che l'utente immetta uno o più dati dalla tastiera. Quando l'elaborazione è completa, questa istruzione memorizza il valore o i valori inseriti in una o più variabili specificate.

```
INPUT "richiesta"; variabili    ' Visualizza un punto di domanda;
                                ' "richiesta" è opzionale.

INPUT "richiesta", variabili    ' Omette il punto di domanda;
                                ' "richiesta" è opzionale.

INPUT; "richiesta"; variabili   ' Il cursore rimane sulla linea
                                ' del prompt dopo l'input.
```

La lista di variabili nell'istruzione INPUT indica il numero e il tipo dei dati che il programma aspetterà di ricevere dalla tastiera quando l'istruzione viene eseguita. La lista può contenere una o più variabili; ogni variabile è separata dalla successiva da una virgola. Per una risposta d'input valida, l'utente deve immettere un valore per ogni variabile in lista ed ogni immissione deve appartenere allo stesso tipo di dati come la variabile corrispondente nella lista d'input. Ogni valore immesso è separato dal successivo da una virgola. Se l'utente commette un errore nell'input, immettendo pochi o troppi dati o immettendo una stringa quando invece è atteso un numero, QuickBASIC visualizza su schermo il seguente messaggio d'errore:

```
Redo from start
```

Dopo questo messaggio, il prompt di INPUT viene visualizzato ancora e l'utente ha un'altra possibilità di immettere una sequenza corretta di valori. Chiaramente è importante per un programma dire all'utente esattamente quali sono i dati da immettere. Questo è il ruolo della richiesta nell'istruzione INPUT. Se presente, questa richiesta opzionale deve essere espressa come un valore stringa costante inserito tra virgolette. (Non è ammessa né una variabile né un'espressione.) Quando l'istruzione INPUT è eseguita, il prompt appare sullo schermo seguito da un cursore luminoso che indica che il programma sta aspettando una risposta dalla tastiera. Tre diversi segni di punteggiatura nell'istruzione INPUT controllano piccoli ma importanti

dettagli del comportamento dell'istruzione. Subito dopo la richiesta possono esserci una virgola o un punto e virgola. Il punto e virgola fa sì che INPUT visualizzi su video un punto di domanda dopo il prompt. In alternativa, una virgola elimina questo punto interrogativo, consentendo di scegliere un qualsiasi segno di punteggiatura come parte del prompt stesso. Un altro punto e virgola può comparire subito dopo la parola chiave INPUT. In risposta a questo punto e virgola, QuickBASIC tiene il cursore dello schermo sulla linea del prompt dopo che l'input è completo. Questo comportamento può rivelarsi utile in programmi che richiedono un controllo attento dell'aspetto linea per linea dello schermo. Senza questo punto e virgola, il cursore si abbassa sulla linea successiva dopo la richiesta quando l'input è completo.

Esempio di INPUT

Il seguente frammento (dal programma *Agenda* presentato nel Capitolo 30), provoca una stringa di istruzioni dall'utente alla tastiera:

```
PRINT "Immmettere l'ora (<7> alle <12> o <1> alle <6>) "
INPUT "per un nuovo appuntamento, o immettere <E> per uscire: ", inVal$
PRINT
IF INSTR(UCASE$(inVal$), "E") <> 0 THEN
   eseguito% = TRUE
ELSE
   inVal% = VAL(inVal$)
   IF inVal% >= 1 AND inVal% <= 12 salvaRevis% = TRUE
```

In questo dialogo d'input, il programma sta chiedendo all'utente di indicare se c'è un'altra linea di appuntamenti da immettere nel calendario per il giorno:

```
Immettere l'ora (<7> alle <12> o <1> alle <6>)
per un nuovo appuntamento, o immettere <E> per uscire: _
```

Se la risposta è affermativa, l'utente immette un numero compreso tra 7 e 12 o tra 1 e 6 per indicare l'ora dell'appuntamento. In caso contrario, digita la lettera Q per terminare il dialogo. Il programma accetta l'input nella variabile di stringa *inVal$*. Ciò consente all'utente di immettere un numero o una lettera. Il programma prima prova a vedere se l'utente ha inserito la lettera E:

```
IF INSTR(UCASE$(inVal$), "E") <> 0 THEN
```

In risposta a questo input, il programma termina il dialogo; altrimenti, cerca di trasformare il valore d'input in un numero e poi prova a vedere se il valore convertito è nell'intervallo orario corretto per un appuntamento:

```
inVal% = VAL(inVal$)
IF inVal% >=1 AND inVal% <= 12 THEN salvaRevis% = TRUE
```

Se è così, il programma continua il dialogo, accettando poi in input una stringa come specificazione dell'appuntamento.

Compatibilità

L'istruzione INPUT in Turbo Basic differisce da quella di QuickBASIC in un dettaglio piccolo, ma significativo: in Turbo Basic, l'utente può tranquillamente immettere un valore stringa in risposta ad un'istruzione INPUT che sta aspettando un numero. Se ciò accade, Turbo Basic semplicemente memorizza il valore zero nella corrispondente variabile numerica. Sullo schermo non compare nessun messaggio d'errore. L'istruzione INPUT in BASICA si comporta come in QuickBASIC.

LINE INPUT

L'istruzione LINE INPUT visualizza su video il prompt opzionale ed aspetta che l'utente immetta una linea di testo. Quando il processo d'immissione è completo, l'istruzione memorizza la linea di testo in una variabile stringa.

```
LINE INPUT "richiesta"; variabile    ' la "richiesta" è opzionale.

LINE INPUT ; "richiesta"; variabile  ' Il cursore rimane sulla linea
                                     ' della richiesta dopo l'input.
```

Il nome della variabile nell'istruzione LINE INPUT corrisponde a una variabile stringa, a lunghezza fissa o variabile. La stringa opzionale di richiesta deve comparire come un valore stringa costante messo tra virgolette. Diversamente dall'istruzione INPUT, LINE INPUT visualizza questa

richiesta sullo schermo senza il punto interrogativo terminale. Se si vuole che la richiesta comprenda qualche particolare segno di punteggiatura, bisogna includerlo all'interno della stringa stessa della richiesta. Il punto e virgola facoltativo subito dopo la parola chiave INPUT, fa sì che QuickBASIC tenga il cursore dello schermo sulla linea della richiesta anche dopo che l'input è completo. Senza il punto e virgola, quando l'input è completo, il cursore si sposta sulla linea sottostante dopo la richiesta. La risposta dell'utente all'istruzione LINE INPUT può contenere caratteri che QuickBASIC normalmente leggerebbe come delimitatori tra input, compresi virgole e punti interrogativi. Senza tener conto di questi caratteri, LINE INPUT memorizza l'intera linea del testo immesso, fino a 255 caratteri, in una variabile stringa specifica.

Un esempio di LINE INPUT

Il programma *Agenda* (presentato nel Capitolo 30) usa LINE INPUT per accettare linee di testo che l'utente immette per descrivere gli appuntamenti. Ecco un esempio:

```
LINE INPUT "12:00 appuntamento di mezzogiorno: "; mattino(12)
```

Grazie a LINE INPUT, l'utente è libero di includere segni di punteggiatura come virgole e punti di domanda in qualsiasi linea di testo.

Compatibilità

LINE INPUT si comporta nello stesso modo sia in Turbo Basic sia in BASICA.

SLEEP

L'istruzione SLEEP crea una pausa di durata prefissata nell'esecuzione di un programma o una pausa che l'utente può terminare premendo un tasto della tastiera.

```
SLEEP secondi     ' Pausa finché i secondi non sono finiti.

SLEEP             ' Pausa finché non viene premuto un tasto.
```

Il valore di *secondi* deve essere un numero positivo all'interno dell'intervallo valido del tipo intero lungo. Se si inserisce questo parametro numerico, SLEEP crea una pausa che dura per il numero specificato di secondi; comunque, la pausa viene interrotta se l'utente preme un tasto o se interviene un evento gestito. Senza il parametro opzionale *secondi*, la pausa viene terminata solo premendo un tasto o con un evento gestito.

Un esempio di SLEEP

Il programma *Agenda* (presentato nel Capitolo 30) usa SLEEP come uno strumento semplice per visualizzare sullo schermo un messaggio fino a che l'utente ha la possibilità di rispondere:

```
PRINT "Premere un tasto qualsiasi per terminare il programma."
SLEEP
```

Questo frammento è parte di una routine di gestione degli errori che prende il controllo nel caso in cui il programma non riesca a trovare nessuno dei file di cui ha bisogno.

Compatibilità

L'istruzione DELAY di Turbo Basic crea una pausa di durata prefissata in un programma, ma non risponde quando viene premuto un tasto della tastiera. BASICA non ha un'istruzione equivalente.

Capitolo 10

Controllo dello schermo e output sulla stampante

Visualizzare delle informazioni sullo schermo in modo chiaro e con attenzione è un aspetto essenziale nei progetti di programmazione più interattivi. Inviare dei dati alla stampante è un'operazione aggiuntiva di output che molti programmi devono eseguire. Questo capitolo descrive le istruzioni QuickBASIC che controllano questi due tipi di output.

OUTPUT DELLO SCHERMO E CONTROLLO

QuickBASIC fornisce molti strumenti per eseguire l'output sullo schermo. Si possono usare queste istruzioni di output per inviare al video singoli valori, per formattare i dati che vi compaiono, per gestire il movimento del cursore e per controllare l'aspetto dello schermo. Le voci di questo capitolo descrivono tutti questi strumenti e ne presentano degli esempi.

CLS

L'istruzione CLS cancella tutta o parte della videata corrente. L'effetto dipende dallo stato del testo o dell'area grafica esistenti.

```
CLS      ' Cancella lo schermo o la finestra attiva.
CLS 0    ' Cancella l'intero schermo, senza badare alle finestre.
CLS 1    ' Cancella un'area grafica o tutto lo schermo grafico.
CLS 2    ' Cancella il testo.
```

L'effetto di CLS effettivamente dipende da tre condizioni: il modo testo corrente, la definizione dell'area grafica o di testo e l'argomento numerico opzionale che si inserisce nella stessa istruzione CLS. (Vedere le voci SCREEN, VIEW PRINT e VIEW per ulteriori dettagli.) Per default, l'istruzione CLS (senza argomento) cancella l'intero schermo e posiziona il cursore nell'angolo in alto a sinistra dello schermo. Se è stata data l'istruzione VIEW PRINT per definire una finestra di testo in SCREEN 0, l'istruzione CLS cancella solo quest'area. Allo stesso modo, se è stata data l'istruzione VIEW per definire un'area grafica su un video, CLS cancella quest'area e lascia intatto il resto dello schermo. CLS 0 cancella sempre l'intero schermo, anche se è stata definita un'area testo o un'area grafica. CLS 1 cancella un'area grafica o l'intero schermo grafico se non è stata definita nessuna area. In modo (SCREEN 0), l'istruzione CLS 1 non ha nessun effetto. CLS 2 cancella l'area testo, come definita dall'istruzione VIEW PRINT in modo testo o modo grafico. Se non è stata definita nessuna area testo, CLS 2 cancella l'intero schermo ad eccezione dell'ultima linea.

Alcuni esempi di CLS

Il programma *Agenda* (presentato nel Capitolo 30) usa CLS in molti diversi contesti. Come molti altri programmi, *Agenda* cancella lo schermo prima di visualizzare qualsiasi informazione:

```
CLS
EsegAgenda me%, gio%, an%
```

Durante un conseguente dialogo d'input, il programma *Agenda* periodicamente deve cancellare molte linee dal fondo dello schermo, mentre tiene visualizzate le informazioni nella zona superiore. Per far ciò, il programma usa CLS più l'istruzione VIEW PRINT:

```
VIEW PRINT 19 TO 23
CLS
PRINT "Immettere l'ora (dalle <7> alle <12> o dalle <1> alle <6>)"
```

```
INPUT "per un nuovo appuntamento, o digitare <Q> per uscire:
      ", inVal$
PRINT
```

Infine, il programma ha bisogno di un modo per cancellare singole linee dallo schermo quando l'utente immette nuovi appuntamenti o riesamina gli appuntamenti esistenti. A questo proposito, il programma contiene una procedura che si chiama *CancLinea*. Questo sottoprogramma riceve un argomento numerico ed usa VIEW PRINT e CLS per cancellare la linea che l'argomento rappresenta:

```
SUB CancLinea (NumLinea%)

' Il sottoprogramma CancLinea cancella una linea
' specifica sul testo dello schermo.

   VIEW PRINT NumLinea% TO NumLinea%
   CLS
   VIEW PRINT

END SUB
```

Quando l'utente riesamina gli appuntamenti, questa procedura aiuta il programma a visualizzare le nuove informazioni sulle opportune linee dello schermo. Ad esempio, ecco due diverse chiamate a *CancLinea:*

```
CancLinea 9
CancLinea inVal% - 3
```

Compatibilità

In Turbo Basic e in BASICA, CLS non ha nessun argomento: l'istruzione cancella o l'intero schermo o l'area grafica corrente.

CSRLIN

La funzione CSRLIN ritorna un intero che rappresenta la posizione verticale corrente del cursore sullo schermo.

```
CSRLIN
```

La funzione non ha argomenti. Il valore di ritorno è un numero di riga compreso da 1 al numero massimo di linee nel corrente modo testo. Il valore 1 rappresenta la linea più in alto del video. Va ricordata anche l'analoga funzione POS che dà invece la posizione corrente orizzontale.

Un esempio di CSRLIN

CSRLIN e POS consentono al programma di registrare una posizione particolare del cursore e poi riportare il cursore nella stessa posizione dopo che sono state eseguite altre operazioni sullo schermo. Il seguente programma dimostra questa tecnica:

```
' Dimostrazione CSRLIN e POS.

CONST FALSE = 0, TRUE = NOT FALSE
CLS : PRINT : PRINT : PRINT : PRINT

PRINT "Pronto per uscire? <S> o <N> ";
DO
   ris$ = UCASE$(INKEY$)
   IF ris$ <> "" THEN
      IF INSTR("SN", ris$) <> THEN
         okRis% = TRUE
         PRINT ris$
      ELSE
         risLinea% = CSRLIN
         risCol% = POS(0)
         LOCATE 24, 10
         PRINT "<S>i o <N>o, prego.";
         SLEEP 1
         LOCATE 24, 10
         PRINT SPACE$(22);
         LOCATE risLinea%, risCol%
      END IF
   END IF
LOOP UNTIL okRis%
```

Questo programma visualizza sullo schermo una richiesta che domanda all'utente una risposta affermativa o negativa. Se l'utente preme un tasto non valido, cioè, un tasto che non sia né S né N, il programma visualizza brevemente un messaggio d'errore in basso sullo schermo e poi ritorna il cursore nella posizione originale. Per realizzare ciò, il programma inizia

registrando la posizione orizzontale e verticale del cursore in due variabili intere:

```
risLinea% = CSRLIN
risCol% = POS(0)
```

Dopo aver visualizzato il messaggio d'errore in fondo sullo schermo, il programma utilizza la seguente istruzione LOCATE per riportare il cursore nella sua posizione originale:

```
LOCATE risLinea%, risCol%
```

Compatibilità

CSRLIN e POS sono disponibili sia in Turbo Basic sia in BASICA.

LOCATE

L'istruzione LOCATE controlla la posizione e l'aspetto del cursore sullo schermo.

```
LOCATE linea, colonna, attDis, margsup, marginf     ' Ogni argomento
                                                    ' è opzionale.
```

Il primo compito dell'istruzione LOCATE è posizionare il cursore in una particolare riga e colonna del video, così che il successivo output sullo schermo inizierà nel punto selezionato. In altre parole, LOCATE consente di controllare l'esatta posizione dei caratteri ASCII che si inviano allo schermo. I primi due argomenti dell'istruzione, *linea* e *colonna*, specificano la posizione. Il numero di linea può essere un intero da 1 (la prima linea dello schermo in alto) fino al numero massimo di linee orizzontali per un particolare modo grafico. (Vedere l'istruzione SCREEN per conoscere le caratteristiche dei modi grafici.) Il numero di colonna è un intero da 1 a 40 o da 1 a 80, secondo la larghezza dello schermo. (Vedere l'istruzione WIDTH.) Inoltre, LOCATE controlla la forma e le dimensioni del cursore lampeggiante. Il terzo argomento, *attDis*, specifica se il cursore comparirà o meno sullo schermo; un valore pari a 0 disattiva il cursore ed il valore 1 lo

attiva. Il quarto ed il quinto argomento, *margsup* e *marginf*, determinano l'altezza del cursore. Questi sono interi che rappresentano le posizioni dei pixel verticali all'interno dell'altezza di una linea orizzontale data. L'intervallo di questi valori dipende dal modo schermo e dall'hardware del video. Ad esempio, un monitor monocromatico in modo testo potrebbe avere un intervallo dell'altezza del pixel che va da 0 a 13. Normalmente il cursore appare come un carattere lampeggiante sottolineato, ma si possono usare *margsup* e *marginf* per aumentare l'altezza in pixel del cursore fino all'altezza totale della linea di testo. In generale, più grande è la differenza tra i valori dati per *margsup* e *marginf* e più alto sarà il cursore. Si può scegliere qualsiasi combinazione di argomenti per LOCATE. Per indicare gli argomenti omessi si utilizzano delle virgole. Ad esempio, la seguente istruzione specifica solo il terzo argomento, per disattivare il cursore:

```
LOCATE , , 0
```

Se si omette il primo argomento, ma si usa il secondo, il cursore comparirà in una nuova posizione nella riga corrente. Viceversa, fornendo il primo argomento, ma non il secondo, il cursore sarà in una nuova posizione nella colonna corrente. L'ultima linea in basso, ad esempio, la linea 25 su uno schermo a 25 linee, è normalmente riservata per scopi specifici, ma, per visualizzare le informazioni sulla linea, un programma può usare l'istruzione LOCATE.

Alcuni esempi di LOCATE

Le seguenti linee prese dal programma *Agenda* (presentato nel Capitolo 30) visualizzano una richiesta su una riga specifica vicino al lato inferiore dello schermo:

```
   LOCATE 23, 5, 1
' ...
   PRINT "Salvo gli appuntamenti di questo giorno? <S> o <N> ";
```

Si noti che il terzo argomento è dato in questa istruzione LOCATE per assicurare che il cursore comparirà sullo schermo. Poiché il quarto ed il quinto argomento del comando LOCATE danno diversi effetti su differenti

schermi e hardware dello schermo, si può decidere di sperimentarli prima di usarli. Un programma come il seguente descrive esattamente le funzionalità di questi argomenti:

```
' Dimostrazione dell'istruzione LOCATE.

PRINT "Test del cursore"
PRINT "---------"
INPUT "Quale modo SCREEN"; modo%

modo% SCREEN

FOR margsup% = 0 TO 13
   FOR marginf% = 0 TO 13
      CLS
      LOCATE 10, 10, 1, margsup%, marginf%
      PRINT USING "margsup = ## marginf = ## ←  "; margsup%; marginf%;
      SLEEP
      DO WHILE INKEY$ <> ""
      LOOP
   NEXT marginf%
NEXT margsup%
```

Questo programma inizia chiedendo un intero per stabilire il modo grafico. Poi il programma gira attraverso diverse combinazioni dei valori *margsup%* e *marginf%* per l'istruzione LOCATE e visualizza su schermo un messaggio che identifica ogni combinazione. (Premere un tasto qualsiasi per visualizzare l'altezza successiva del cursore.) Quando si attiva questo programma, su alcuni video, si può notare che il cursore si divide in due parti se il valore di *margsup%* è maggiore del valore di *marginf%*. Gli argomenti *attDis, margsup* e *marginf* non hanno alcun effetto in alcuni modi grafici.

Compatibilità

Sia Turbo Basic sia BASICA supportano l'istruzione LOCATE.

POS

La funzione POS ritorna un intero che rappresenta la posizione orizzontale corrente del cursore sullo schermo.

```
POS(0)
```

La funzione accetta un argomento fittizio pari a zero. Il valore di ritorno è un numero della colonna cha va da 1 al numero massimo di colonne nel modo grafico corrente (40 o 80). Il valore 1 rappresenta la prima colonna a sinistra del video. La funzione CSRLIN dà la posizione verticale corrente del cursore.

Un esempio di POS

POS e CSRLIN consentono al programma di registrare una particolare posizione del cursore per poi rimetterlo nella stessa posizione dopo che sono state eseguite altre operazioni sullo schermo. Il programma dimostrativo presentato nella voce CSRLIN illustra questa tecnica. Il programma registra una particolare posizione del cursore in due variabili intere:

```
risLinea% = CSRLIN
risCol% = POS(0)
```

Dopo che sono state eseguite altre operazioni sullo schermo, il programma usa la seguente istruzione LOCATE per rimettere il cursore nella sua posizione originale:

```
LOCATE risLinea%, risCol%
```

Compatibilità

POS e CSRLIN sono disponibili sia in Turbo Basic sia in BASICA.

PRINT

L'istruzione PRINT invia informazioni allo schermo.

```
PRINT                        ' Stampa una linea vuota.

PRINT valore; ...; valore;   ' Visualizza i valori in posizioni
```

```
                             ' adiacenti dello schermo.
                             ' Il punto e virgola finale è opzionale.

PRINT valore, ..., valore,   ' Visualizza i valori in diverse
                             ' aree di stampa.
                             ' La virgola finale è opzionale.
```

La lista di dati nell'istruzione PRINT può comprendere valori stringa o numerici, espressi come costanti, variabili o espressioni. Se si inserisce un'espressione nella lista, PRINT valuta l'espressione e visualizza sullo schermo il valore risultante. Un'istruzione PRINT che non contiene nessuna lista di dati stampa una linea vuota. La punteggiatura determina la spaziatura tra i valori emessi; ad esempio il punto e virgola sistema i valori fianco a fianco senza spazio bianco. (Comunque, un numero positivo è sempre preceduto e seguito da uno spazio. Un numero negativo è seguito, ma non preceduto da uno spazio.) Se si mette un punto e virgola dopo l'ultimo valore nella lista PRINT, il successivo PRINT inizia visualizzando i dati seguenti al primo valore. Senza un punto e virgola finale, l'istruzione PRINT invia automaticamente allo schermo una combinazione di ritorno del carrello e salto di riga, così che ogni nuova istruzione PRINT inizia visualizzando informazioni su una nuova linea. Al contrario, le virgole nella lista PRINT fanno sì che QuickBASIC visualizzi ogni valore all'inizio della successiva *area di stampa*. Le aree di stampa sono aree di tabulazione di 14 spazi che iniziano alle colonne 1, 15, 29, 43 e 57 su uno schermo a 80 colonne. Una virgola messa dopo l'ultimo valore di un'istruzione PRINT significa che l'istruzione PRINT successiva inizierà visualizzando i dati all'inizio dell'area di stampa successiva. Per visualizzare numeri reali, l'istruzione PRINT può scegliere il formato a virgola fissa o quello a virgola mobile. (Consultare la voce "Tipi a Virgola Mobile" nel Capitolo 1 per informazioni su questi formati.) Il formato a virgola fissa è utilizzato, se è possibile farlo, rispettando i limiti della precisione del tipo dichiarato, ovvero 7 cifre per valori a precisione semplice e 15 cifre per quelli a doppia precisione. Diversamente, PRINT visualizza un numero nel formato a virgola mobile, usando la lettera E o D per contrassegnare l'esponente. (Per controllare il formato dei numeri visualizzati sullo schermo, si può usare invece l'istruzione PRINT USING.)

Alcuni esempi di PRINT

Le seguenti istruzioni PRINT sono state prese dal programma *Compleanno*, presentato nel Capitolo 28. Il primo esempio visualizza sullo schermo tre stringhe:

```
PRINT TITLE; RIGHT$(DATE$, 4); "."
```

In questa lista PRINT, la prima stringa è rappresentata come un nome costante simbolico, la seconda è un'espressione che consiste in chiamate di funzione e la terza è un valore stringa literal. Le istruzioni successive mostrano come il punto e virgola può essere usato per eliminare il ritorno del carrello tra un'istruzione PRINT e la successiva:

```
PRINT giorSett$(NumGiorno%(me%, gio%, an%)); "., ";
PRINT nomiMese$(me%); ". "; LTRIM$(STR$(gio%))
```

Il risultato di queste due linee è una sola stringa della data, come:

```
Sab., Mar 18
```

Infine, la sequenza successiva indica l'uso dell'istruzione PRINT priva di una lista di dati, utilizzata per avere una linea vuota tra due linee di output.

```
PRINT "Organizza la tabella:"
PRINT
PRINT " N)omi."
```

Compatibilità

L'istruzione PRINT è uguale sia in Turbo Basic sia in BASICA. L'interprete BASICA consente l'uso del punto interrogativo (?) come un'abbreviazione per l'istruzione PRINT. Si può usare questa abbreviazione anche in QuickBASIC ma l'editor automaticamente converte il punto di domanda nella parola chiave PRINT.

PRINT USING

L'istruzione PRINT USING visualizza su schermo i dati formattati.

```
PRINT USING formato; valore; ...; valore;  ' Il punto e virgola
                                           ' finale è opzionale.
```

PRINT USING è un'istruzione versatile che dà il controllo sul modo in cui i dati vengono visualizzati su schermo. Si assegnano le istruzioni relative alle modalità di output in una stringa *speciale di formato*, che si trova subito dopo la parola chiave USING. La stringa di formato può presentarsi come un valore stringa literal messo tra virgolette, come una variabile o un'espressione di tipo stringa. Nella punteggiatura dell'istruzione PRINT USING un punto e virgola segue sempre la stringa di formato. La stringa di formato è costituita da speciali caratteri o set di caratteri designati che indicano l'aspetto desiderato dei dati in uscita. Ecco un riassunto dei caratteri disponibili in una stringa di formato:

#	Rappresenta una singola cifra numerica
.	Indica la posizione del punto decimale in un numero
,	Visualizza la virgola alla sinistra del punto decimale
$$	Colloca il simbolo dollaro alla sinistra del numero
**	Riempie con degli asterischi le posizioni a sinistra della cifra più significativa di un numero
**$	Visualizza gli asterischi come sopra, e poi un simbolo dollaro
+	Visualizza il segno di un numero
-	Visualizza un meno dopo un numero negativo
^^^^	Visualizza un numero nel formato a virgola mobile
_	(Underscore) segnala un carattere literal in una stringa di formato
&	Visualizza un valore stringa
\\	Visualizza parte di un valore stringa (lunghezza degli spazi interposti più 2)
!	Visualizza solo il primo carattere di un valore stringa

Una stringa di formato contiene un "template", cioè un gruppo di caratteri che individuano il formato, per ogni variabile che compare nella lista di output dell'istruzione PRINT USING. L'ordine dei template corrisponde all'ordine dei valori d'output nella lista dell'istruzione. Inoltre, la stringa di formato può contenere del testo che sarà visualizzato direttamente su video con i dati. I valori nella lista dati PRINT USING possono essere costanti, variabili o espressioni. Ognuno è separato dal successivo da un punto e virgola. (Mentre si sta scrivendo un'istruzione PRINT USING, quando si completa la linea, si possono aggiungere delle virgole ai punti e virgola.) Per eliminare l'andata a capo (cr-lf) che normalmente si verifica dopo un'istruzione PRINT USING, si può mettere un punto e virgola alla fine dell'istruzione. In questo caso, l'istruzione PRINT o PRINT USING successiva inizierà il suo output sulla stessa linea dove l'istruzione precedente lo aveva terminato. Si verifica un errore run-time se un template è inadatto al tipo di dati del corrispondente valore d'output. Ad esempio, se la stringa di formato è preparata per un valore numerico ed il valore corrispondente nella lista di dati PRINT USING è una stringa, QuickBASIC visualizza il messaggio d'errore "Type mismatch".

Alcuni esempi di PRINT USING

Il programma *Agenda* (presentato nel Capitolo 30) contiene alcuni interessanti esempi di PRINT USING per visualizzare linee di output per gli appuntamenti del giorno. Ad esempio, si considerino queste linee:

```
PRINT USING "##:0 a.m. > "; ora%;
PRINT matt(ora%)
```

L'istruzione iniziale PRINT USING visualizza semplicemente l'ora. Si noti che i caratteri "##" nella stringa di formato corrispondono ad un valore numerico memorizzato nella variabile *ora%*. L'istruzione PRINT seguente visualizza il testo dal vettore *matt(ora%)* sulla stessa linea. Ecco un esempio dell'output che queste due istruzioni causano:

```
9:00 a.m. > Incontro con M.M.C. nella sala congressi principale.
```

Il programma *Compleanno* (listato nel Capitolo 28) contiene ulteriori esempi di PRINT USING. Ad esempio, il frammento seguente visualizza sullo

schermo tre numeri formattati come parte dell'output di tabulazione del programma:

```
PRINT USING " ##.# "; staff(i%).Anzian;
' ...
PRINT USING "$$###.## "; gratif%;
PRINT USING "## "; staff(i%).etàAtt;
```

Queste istruzioni visualizzano tre valori numerici in ogni record della tabella d'output presa dal programma *Compleanno*, gli anni per cui un impiegato ha lavorato per la ditta, la gratifica annuale dell'impiegato e la sua età. Ad esempio, ecco due record della risultante tabella d'output:

```
Nome           Qualifica   Anzianità   Gratifica      Età   Compleanno

Billi, Anna    Segretaria     2,9      100.000        24    Ven., Apr. 6
Rossi, Paolo   Impiegato      5,7      250.000        35    Sab., Lug. 7
```

Le due istruzioni finali PRINT USING in questo frammento del programma potrebbero essere unite in una sola istruzione, come segue:

```
PRINT USING "$$###.## ## "; gratif%; staff(i%).etàAtt;
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione PRINT USING.

SPC

SPC è una funzione che ordina all'istruzione PRINT di muovere il cursore in avanti, dopo un numero specifico di spazi.

```
PRINT SPC(numero); valore; ...
```

La funzione SPC va usata nel contesto di un'istruzione PRINT, LPRINT o PRINT#. Come risultato di SPC, un comando output salta il numero di spazi specificato nell'argomento della funzione.

Un esempio di SPC

Il seguente esempio mostra come SPC possa essere usato per aiutare a creare tabelle di colonne allineate:

```
' Dimostrazione della funzione SPC.

DIM vendite(1988 TO 1990), totali(1988 TO 1990)

CLS

PRINT SPC(11); "Vendite Primo Trimestre"
PRINT SPC(14); "1988 al 1990"
PRINT
PRINT SPC (9);
FOR anno% = 1988 TO 1990
   PRINT anno%; SPC(6);
NEXT anno%
PRINT : PRINT

FOR mese% = 1 TO 3
   READ me$
   PRINT me$; SPC(5);
   FOR anno% = 1988 TO 1990
      READ vendite(anno%)
      totali(anno%) = totali(anno%) + vendite(anno%)
      PRINT USING "$##,###"; vendite(anno%); SPC(5);
   NEXT anno%
   PRINT
NEXT mese%

PRINT
PRINT "Totale ";
FOR anno% = 1988 TO 1990
   PRINT USING "$##,###"; totali(anno%); SPC(5);
NEXT anno%

DATA Gen, 19.000.000, 22.000.000, 20.000.000
DATA Feb, 12.500.000, 19.600.000, 17.200.000
DATA Mar, 18.200.000, 16.500.000, 15.400.000
```

Il programma legge alcune informazioni da un set di istruzioni DATA e poi visualizza su schermo le informazioni. La maggior parte delle istruzioni PRINT e PRINT USING nel programma usano la funzione SPC per formare colonne di dati. Ecco l'output del programma:

```
           Vendite Primo Trimestre
              1988 al 1990

           1988        1989        1990

Gen      19.000.000  22.000.000  20.000.000
Feb      12.500.000  19.600.000  17.200.000
Mar      18.200.000  16.500.000  15.400.000

Totale   49.700.000  58.100.000  52.600.000
```

Compatibilità

SPC è uguale sia in Turbo Basic sia in BASICA.

TAB

TAB è una funzione che ordina all'istruzione PRINT di spostare il cursore in avanti ad un numero specifico di colonna.

```
PRINT TAB(numero); valore; ...
```

La funzione TAB si usa nel contesto di un'istruzione PRINT, LPRINT o PRINT#. Come risultato di TAB, un comando d'output si sposta in avanti con una tabulazione fino al numero di colonna fornito come argomento della funzione.

Un esempio di TAB

Il seguente esempio mostra come TAB può essere usato per creare delle tabelle con colonne allineate:

```
' Dimostrazione della funzione TAB.

DIM vendite(1988 TO 1990)
DIM totali(1988 TO 1990)
DIM tabx%(1988 TO 1990)

FOR colonna% = 1988 TO 1990
```

```
   READ tabx%(colonna%)
NEXT colonna%

CLS

PRINT TAB(11); "Vendite Primo Trimestre"
PRINT TAB(14); "1988 al 1990"
PRINT

FOR anno% = 1988 TO 1990
   PRINT TAB(tabx%(anno%)); anno%;
NEXT anno%
PRINT : PRINT

FOR mese% = 1 TO 3
   READ me$
   PRINT me$;
   FOR anno% = 1988 TO 1990
      READ vendite(anno%)
      totali(anno%) = totali(anno%) + vendite(anno%)
      PRINT USING "$##,###"; TAB(tabx%(anno%)); vendite(anno%);
   NEXT anno%
   PRINT
NEXT mese%
PRINT
PRINT "Totale ";
FOR anno% = 1988 TO 1990
   PRINT USING "$##,###"; TAB(tabx%(anno%)); totali(anno%);
NEXT anno%

DATA 10, 20, 30
DATA Gen, 19.000.000, 22.000.000, 20.000.000
DATA Feb, 12.500.000, 19.600.000, 17.200.000
DATA Mar, 18.200.000, 16.500.000, 15.400.000
```

Questo programma utilizza il vettore intero *tabx* per memorizzare le tabulazioni per la tabella di output. Le istruzioni PRINT del programma inviano gli elementi di questo vettore come argomenti alla funzione TAB per spostarsi in avanti con una tabulazione su particolari colonne dello schermo. Ecco l'output del programma:

```
           Vendite Primo Trimestre
                1988 al 1990

           1988        1989        1990

Gen      19.000.000  22.000.000  20.000.000
Feb      12.500.000  19.600.000  17.200.000
Mar      18.200.000  16.500.000  15.400.000
```

```
Totale   49.700.000  58.100.000  52.600.000
```

Un'altra versione di questo programma si trova alla voce SPC.

Compatibilità

Sia Turbo Basic sia BASICA supportano la funzione TAB per utilizzare le istruzioni PRINT.

VIEW PRINT

L'istruzione VIEW PRINT stabilisce una *finestra di testo*, ossia un'area dello schermo su cui successivamente verranno eseguiti alcuni comandi output.

```
VIEW PRINT margsup TO marginf      ' Stabilisce una finestra di testo.

VIEW PRINT                         ' Ripristina l'intero schermo come
                                   ' finestra di testo.
```

Gli argomenti *margsup* e *marginf* sono interi che rappresentano i numeri di linea sullo schermo. L'intervallo possibile per questi valori va da 1 fino al numero delle righe del modo schermo corrente. Il valore di *margsup* deve essere minore o uguale al valore di *marginf*. La finestra risultante è l'area dello schermo da *margsup* a *marginf* compresi. Quando l'area è attiva, molte istruzioni di output operano solo all'interno della finestra:

- L'istruzione CLS cancella la finestra di testo e lascia il resto dello schermo intatto.
- L'istruzione PRINT visualizza delle informazioni e, se necessario, le fa scorrere all'interno della finestra.
- L'istruzione LOCATE accetta solo quegli indirizzi di riga che si trovano nella finestra. (Un'istruzione LOCATE con un indirizzo di riga fuori dalla finestra dà luogo ad un errore run-time.)

Per ripristinare l'intero schermo come finestra di testo, bisogna usare VIEW PRINT senza argomenti.

Esempi di VIEW PRINT

Il programma *Agenda* (presentato nel Capitolo 30) usa VIEW PRINT in molti contesti per controllare le attività dello schermo. Ad esempio, il seguente frammento cancella un'area vicina al margine inferiore del video e vi visualizza un prompt, senza influenzare i contenuti del resto dello schermo:

```
VIEW PRINT 19 TO 23
CLS
PRINT "Immettere l'ora (<7> alle <12> o <1> alle <6>) "
INPUT "per un nuovo appuntamento, oppure imettere <Q> per uscire:
     ", inVal$
PRINT
```

Il programma *Agenda* contiene anche una funzione che si chiama *CancLinea*, il cui compito è cancellare i contenuti di una sola linea dello schermo:

```
SUB CancLinea (NumLinea%)

' Il sottoprogramma CancLinea cancella una linea
' specifica dal testo.

   VIEW PRINT NumLinea% TO NumLinea%
   CLS
   VIEW PRINT

END SUB
```

CancLinea crea una finestra di testo ad una linea, la cancella e poi ripristina come finestra l'intero schermo. Per ulteriori notizie su questa procedura si può consultare la voce CLS.

Compatibilità

Il Turbo Basic non ha l'istruzione VIEW PRINT, ma supporta l'istruzione VIEW per definire le aree grafiche. Le versioni più recenti di BASICA (e GW-BASIC) supportano VIEW PRINT.

WRITE

L'istruzione WRITE visualizza su schermo delle informazioni.

```
WRITE                         ' Risulta in una linea vuota.

WRITE valore, ..., valore     ' Visualizza i valori sullo schermo.
```

La lista di dati nell'istruzione WRITE può comprendere valori stringa o numerici, espressi come costanti, variabili o espressioni. Se si inserisce un'espressione nella lista, WRITE valuta l'espressione e visualizza su video il valore risultante. Un'istruzione WRITE che non contiene nessuna lista di dati risulta in una linea vuota. A differenza dell'istruzione PRINT, WRITE separa ogni output dal successivo con una virgola che è visualizzata su schermo. Inoltre, WRITE mette i valori stringa tra virgolette. WRITE#, l'analogo dell'istruzione WRITE per operare con i file, è più comunemente usato della versione per lo schermo.

Un esempio di WRITE

Il seguente breve esercizio illustra semplicemente il tipo di output che WRITE dà:

```
numero1= 10000
numero2 = 25
parola$ = "Output"

WRITE numero1, numero2, parola$, numero1 / numero2
```

Ecco l'output da questa istruzione WRITE:

```
10000,25,"Output",400
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione WRITE.

PRINTER OUTPUT

Per comodità di programmazione, QuickBASIC fornisce due istruzioni incorporate che inviano informazioni direttamente alla stampante, LPRINT e LPRINT USING. Specificatamente, queste istruzioni lavorano con il dispositivo di output designato LPT1. Inoltre, la funzione incorporata LPOS può aiutare a controllare l'attività della stampante.

LPOS

La funzione LPOS ritorna un intero che rappresenta la posizione corrente di stampa sulla riga.

```
LPOS(stampante)
```

L'argomento di LPOS è un intero positivo che rappresenta una stampante: 1 per LPT1, 2 per LPT2 o 3 per LPT3. Il valore di ritorno è un intero che può andare da 1 alla larghezza massima di stampa della stampante, 80 per default.

Un esempio di LPOS

Il seguente programma dimostra l'uso della funzione LPOS per controllare la larghezza di ogni linea che è inviata alla stampante:

```
' Dimostrazione della funzione LPOS.

larghMass% = 25
READ parole%

LPRINT
FOR i% = 1 TO parole%
   READ parola$
   LPRINT parola$; " ";
   IF LPOS(1) >= larghMass% THEN LPRINT
NEXT i%
LPRINT
```

```
DATA 12
DATA La, funzione, LPOS, ritorna, un, intero, che rappresenta
DATA la, posizione, corrente, orizzonatale, di, stampa.
```

Il programma legge una serie di parole dalle istruzioni DATA ed invia le parole alla stampante sotto forma di frase. Ogni volta che la posizione della stampante passa il margine destro di *larghMass%*, il programma invia alla stampante un segnale di ritorno del carrello:

```
IF LPOS(1) >= larghMass% THEN LPRINT
```

Con *larghMass%* fissata a 25, ecco l'output sulla stampante eseguito dal programma:

```
La funzione LPOS ritorna
un intero che rappresenta
la posizione corrente
orizzontale di stampa.
```

Compatibilità

Sia Turbo Basic sia BASICA supportano la funzione LPOS.

LPRINT E LPRINT USING

Le istruzioni LPRINT inviano informazioni alla stampante.

```
LPRINT                      ' Risulta in una linea vuota.

LPRINT valore; ...; valore;  ' Visualizza i valori su posizioni
                             ' adiacenti dello schermo. Il punto e
                             ' virgola finale è opzionale.

LPRINT valore, ..., valore,  ' Visualizza i valori in aree di stampa.
                             ' La virgola finale è opzionale.
LPRINT USING formato; valore; ...; valore;    ' Stampa i valori
                                              ' formattati.
```

Le istruzioni LPRINT mandano informazioni alla stampante LPT1. LPRINT e LPRINT USING hanno tutte le stesse opzioni dei loro analoghi PRINT e PRINT USING per l'output sullo schermo. La lista di dati nelle

istruzioni LPRINT può comprendere valori stringa o numerici, espressi come costanti, variabili o espressioni. Un'istruzione LPRINT che non contiene nessuna lista di dati risulta in una linea vuota. La punteggiatura nell'istruzione LPRINT determina la spaziatura tra i valori d'output. I punti e virgola mettono i valori fianco a fianco senza nessuno spazio extra; le virgole fanno sì che QuickBASIC stampi i valori in aree di stampa di 14 spazi. Un punto e virgola o una virgola messi alla fine di un'istruzione LPRINT o LPRINT USING elimina il ritorno del carrello e il salto di riga che normalmente viene eseguito alla fine di ogni linea stampata. LPRINT USING riceve come argomento una *stringa di formato*, costituita da speciali caratteri o set di caratteri che descrivono l'aspetto desiderato dei dati emessi. Per ulteriori dettagli su tutte queste opzioni, consultare le voci PRINT e PRINT USING.

Alcuni esempi di LPRINT

Il programma *Mese*, presentato nel Capitolo 29, dà all'utente la possibilità di visualizzare su schermo i mesi del calendario o inviarli alla stampante. Una procedura chiamata *StampMese* prende il controllo della stampa di informazioni se l'utente seleziona questa opzione. La procedura inizia stampando il nome del mese e le abbreviazioni dei giorni:

```
LPRINT SPACE$(12 - LEN(strMe$) \ 2); strMe$; anno%
LPRINT
LPRINT " D L M M G V S"
LPRINT " "; STRING$(26, "-")
```

Poi, all'interno di un paio di cicli, la procedura usa l'istruzione LPRINT USING per stampare ogni data nella corrente posizione della colonna:

```
LPRINT USING " ##"; dataCorr%;
```

Compatibilità

Sia Turbo Basic sia BASICA supportano l'istruzione LPRINT.

Capitolo 11

Input e output su disco: stile QuickBASIC

La gestione dei file di dati è molto più facile e più chiara nelle ultime versioni di QuickBASIC di quanto non lo fosse nelle prime versioni del linguaggio BASIC. La prima innovazione di QuickBASIC in questo contesto è il supporto per le strutture record definite dall'utente: ora si può usare una variabile record per definire la struttura di un file ad accesso random. Quando si eseguono operazioni di lettura o scrittura sui file di dati, QuickBASIC automaticamente gestisce gli assegnamenti necessari e formatta le conversioni per i singoli campi di dati, compiti che si devono eseguire manualmente quando si devono seguire dettagliatamente i vari passi della programmazione. Si può lavorare con tre tipi di file in QuickBASIC. Un file *sequenziale* generalmente viene letto secondo un ordine preciso: dall'inizio alla fine. Al contrario, un file ad *accesso random* consente di accedere direttamente ad ogni record per operazioni di lettura o scrittura. Infine, QuickBASIC permette il modo ad accesso *binario*, per leggere o scrivere file a livello del singolo byte. Il comando OPEN inizia le operazioni di input ed output per tutti e tre i tipi di accesso ai file. Un file sequenziale può essere aperto in uno qualsiasi dei tre modi: INPUT, OUTPUT o APPEND. Le istruzioni che leggono o scrivono un file sequenziale sono simili a quelle che gestiscono gli input dalla tastiera e gli output sullo schermo: l'istruzione INPUT# legge singoli dati da un file sequenziale e l'istruzione LINE INPUT # legge intere linee alla volta. (In

entrambi i casi, un programma può usare la funzione EOF per controllare ripetutamente se si è giunti alla fine del file durante un processo di lettura.) Per scrivere dati in un file sequenziale, si può scegliere tra le istruzioni WRITE#, PRINT# o PRINT# USING, a seconda delle necessità di formattazione dati del programma. Per accedere ad un file random si usano speciali istruzioni d'input e output. L'istruzione GET# si muove direttamente su un particolare record nel file, legge i dati e li copia in una variabile record. Inoltre, l'istruzione PUT# scrive un record di dati in una specifica posizione nel file. (In alternativa, si può usare il comando SEEK per selezionare la posizione di un record nel file prima di eseguire un'operazione di lettura o scrittura.) La difficoltà maggiore nella gestione di un database ad accesso random è mantenere traccia di dove si trovano i singoli record nel file. Una soluzione comune è creare un file indice separato che contenga una lista di singoli campi, detti *chiavi*, che identificano ogni record nel database. Nell'indice ad ogni chiave è accoppiato un intero che rappresenta la posizione del corrispondente record nel file ad accesso random. Un programma può usare questo indice per trovare un record esistente, circa nello stesso modo in cui si utilizza l'indice di un libro per cercare un particolare argomento. Questa tecnica richiede alcuni speciali strumenti: una routine che organizza l'indice aggiungendovi le chiavi, quando vengono aggiunti nuovi record al database e all'indice; e una routine di ricerca che può localizzare una specifica chiave velocemente senza dover consultare tutto l'indice. Il programma *Agenda*, presentato nel Capitolo 30, illustra tutti i dettagli di questa tecnica ad indice. Oltre ai modi ad accesso random e sequenziale, QuickBASIC consente di aprire un file nel modo BINARY, per leggere o scrivere singoli byte. Per questo modo le istruzioni GET# e PUT# sono i principali strumenti d'input e output, ma in questo caso prendono in considerazione singoli byte piuttosto che record strutturati. Oltre alle istruzioni che eseguono specifiche operazioni di lettura e scrittura, QuickBASIC fornisce una collezione di funzioni speciali che danno informazioni su un file aperto. Questi strumenti comprendono EOF ("la funzione fine file"), LOF ("lunghezza del file"), LOC e SEEK (per identificare la posizione corrente dei dati in un file) e FILEATTR (per identificare il modo di accesso al file). Infine, i comandi CLOSE e RESET terminano le operazioni d'input ed output per un particolare file o per tutti i file aperti. Le voci di questo capitolo danno dettagli ed esempi di tutte queste istruzioni e funzioni per manipolare i file di dati.

CLOSE

L'istruzione CLOSE termina l'operazione d'input e/o output iniziata da una o più istruzioni OPEN.

```
CLOSE

CLOSE #num, ...    ' Possono essere listati molti numeri di file.
                   ' Il simbolo # è facoltativo.
```

Per chiudere uno o più file, si può immettere l'istruzione CLOSE con uno o più argomenti *#num*. Ogni argomento si riferisce al numero assegnato ad un file da una precedente istruzione OPEN. In alternativa, l'istruzione CLOSE inserita senza argomenti chiude tutti i file aperti. Una volta che il file è stato chiuso, il numero del file è disponibile per essere riutilizzato.

Alcuni esempi di CLOSE

Il programma *Agenda*, listato nel Capitolo 30, potenzialmente apre due file durante un'esecuzione. Il file primario è AGENDA.DAT, un file ad accesso random che contiene un database degli appuntamenti. Il file secondario è AGENDA.NDX, un file indice che il programma usa per sistemare specifici record all'interno del file del database. Il programma assegna al database il numero 1 come numero del file:

```
OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = LEN (giorno)
```

Questo file rimane aperto essenzialmente per l'intera esecuzione del programma e viene chiuso solo quando il programma ha completato tutte le operazioni con il file:

```
CLOSE #1
```

Poiché un file ad accesso random è disponibile per scrivere e per leggere contemporaneamente, non serve chiudere il file e riaprirlo tra due operazioni di diverso tipo. Al contrario, il programma *Agenda* apre il file indice solo in quelle occasioni in cui il file serve. Quando una data operazione è completata, il file viene immediatamente richiuso. Ad esempio, ecco il frammento che apre il file indice e vi salva una lista ordinata:

```
OPEN "AGENDA.NDX" FOR OUTPUT AS #2

FOR imm% = 1 TO contImm%
WRITE #2, indData(imm%).datalungh, indData(imm%).posData
NEXT imm%

CLOSE #2
```

Un file sequenziale è aperto solo per un'operazione alla volta, per scrivere o per leggere, ma non per entrambe. Per eseguire entrambe le operazioni, il programma *Agenda* deve chiudere il file indice sequenziale dopo averlo letto e poi riaprirlo per scrivere di nuovo.

Compatibilità

Sia Turbo Basic sia BASICA supportano l'istruzione CLOSE.

EOF

La funzione EOF ritorna true quando si incontra la fine di un file aperto.

```
EOF(num)
```

L'argomento *num* è il numero del file assegnatogli da una precedente operazione OPEN. Durante la lettura di un file sequenziale, EOF ritorna false finché ci sono ancora dei dati da leggere. La funzione diventa vera dopo che l'ultimo valore è stato letto. Per un file ad accesso random, EOF è vero se un intero record non può essere letto dal file. Si può usare EOF anche con un file di comunicazione.

Alcuni esempi di EOF

La funzione EOF consente ad un programma di leggere un file dall'inizio alla fine senza nessuna ulteriore informazione sulla lunghezza del file. La funzione EOF appare spesso come la condizione di un ciclo che legge i dati da un file; quando EOF diventa vera, il ciclo si ferma. Il seguente program-

ma, chiamato *Linee*, illustra EOF. Il programma conta tutte le linee non vuote contenute in un file di testo:

```
' LINEE.BAS
' Conta il numero di linee non vuote in un file di testo.

CONST FALSE = 0, TRUE = NOT FALSE

noTrov% = FALSE
nomeFile$ = COMMAND$

' Tentativo di aprire il file dell'utente.

ON ERROR GOTO noQueFile
   OPEN nomeFile$ FOR INPUT AS #1
ON ERROR GOTO 0

' Terminare il programma se non si trova il file.

IF noTrov% THEN END

' Altrimenti, contare le linee non vuote nel file.

ContLin% = 0
DO WHILE NOT EOF(1)
   LINE INPUT #1, inLine$
   IF LTRIM$(inLin$) <> "" THEN ContLin% = ContLin% + 1
LOOP

CLOSE #1

' Visualizzare i risultati.

PRINT

IF ContLin% = 1 THEN
   PRINT nomeFile$; " contiene 1 linea di testo.";
ELSE
   PRINT nomeFile$; " contiene"
   PRINT USING "#.### linee di testo."; ContLin%
END IF

END

' Gestione degli errori per file scomparso.

noQueFile:
noTrov% = TRUE
 PRINT
```

```
   PRINT "Non si trova questo file sul disco."
   noTrov% = TRUE
RESUME NEXT
```

Il programma *Linee* è progettato come uno strumento del DOS, per essere compilato e attivato direttamente dal prompt del DOS. Si può inviare al programma il nome di un file di testo memorizzato su disco. Il programma apre il file, conta le linee non vuote e riporta il risultato sullo schermo. Ad esempio, ecco come l'esecuzione potrebbe procedere per contare le linee di un testo chiamato DATA.TXT:

```
C>LINEE DATA.TXT

DATA.TXT contiene
   39 linee di testo.
```

Dopo aver aperto il file in questione (con 1 come numero di file), il programma *Linee* usa il seguente ciclo DO per leggere e contare le linee:

```
DO WHILE NOT EOF(1)
   LINE INPUT #1, inLin%
   IF LTRIM$(inLin$) <> "" THEN ContLin% = ContLin% + 1
LOOP
```

Si noti che una chiamata alla funzione EOF appare come condizione della clausola WHILE in questo ciclo. Il ciclo legge il file linea per linea ed incrementa il valore di *ContLin%* per ogni linea non vuota. Il conteggio è considerato completo quando EOF ritorna un valore true.

Compatibilità

Sia Turbo Basic sia BASICA supportano questa importante funzione.

FILEATTR

La funzione FILEATTR dà due informazioni a proposito di un file di dati aperto: il modo in cui è aperto ed il numero del descrittore del file del DOS. Entrambe queste voci vengono fornite come interi.

```
FILEATTR(num, selez)    ' selez è 1 per il modo o 2 per
                        ' il descrittore del DOS.
```

L'argomento *num* è il numero assegnato al file da una precedente istruzione OPEN. Il secondo argomento sceglie una delle due informazioni che dà FILEATTR: un argomento pari ad 1 risulta in un intero che rappresenta il modo in cui il file è aperto; e un argomento pari a 2 dà il descrittore del file del DOS. Se si attribuisce 1 all'argomento *selez*, FILEATTR ritorna uno dei seguenti interi per rappresentare il modo del file:

1	Il modo INPUT sequenziale
2	Il modo OUTPUT sequenziale
4	Il modo RANDOM
8	Il modo APPEND sequenziale
32	Il modo BINARY

Un esempio di FILEATTR

Il seguente esercizio illustra l'utilizzo di FILEATTR:

```
OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = 600
OPEN "AGENDA.NDX" FOR INPUT AS #2
OPEN "PROVA.DAT" FOR BINARY AS #3

PRINT "AGENDA.DAT" è aperto in modo "; FILEATTR(1, 1)
PRINT "AGENDA.NDX" è aperto in modo "; FILEATTR(2, 1)
PRINT "PROVA.DAT" è aperto in modo "; FILEATTR(3, 1)

RESET
```

Questo programma apre tre file e poi visualizza i codici che rappresentano i modi in cui i file sono aperti. Ecco l'output del programma:

```
AGENDA.DAT è aperto in modo 4
AGENDA.NDX è aperto in modo 1
PROVA.DAT è aperto in modo 32
```

Un programma che esegue complesse gestioni di file, che apre altri file in vari modi, può usare FILEATTR per mantenere traccia di ogni stato del file.

Compatibilità

FILEATTR non è disponibile né in Turbo Basic né in BASICA.

FREEFILE

La funzione FREEFILE ritorna il numero di file successivo che si può usare durante un'esecuzione del programma.

```
FREEFILE
```

La funzione non ha argomenti. Il valore di ritorno è un intero compreso tra 1 e 255, che rappresenta il numero più piccolo di file che non è in uso per un file aperto. Se un programma utilizza numeri di file non contigui, FREEFILE ritorna il numero più piccolo disponibile. Ad esempio, se si stanno usando i numeri di file 1, 4 e 5, FREEFILE ritorna il numero 2.

Un esempio di FREEFILE

Questo esercizio descrive l'utilizzo di FREEFILE:

```
OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = 600
OPEN "AGENDA.NDX" FOR INPUT AS #2
OPEN "PROVA.DAT" FOR BINARY AS #3

PRINT
PRINT "Il successivo numero di file disponibile è "; FREEFILE

RESET

PRINT
PRINT "Dopo RESET:"
PRINT "Il successivo numero di file disponibile è "; FREEFILE
```

Il programma apre tre file e visualizza il valore di ritorno di FREEFILE. Poi il programma esegue un'istruzione RESET per chiudere tutti i file e chiama ancora FREEFILE. Ecco l'output del programma:

```
Il successivo numero di file disponibile è 4
```

```
Dopo RESET:
Il successivo numero di file disponibile è 1
```

Un programma che esegue complesse gestioni di file con più file aperti può usare FREEFILE per mantenere una traccia dei numeri di file disponibili.

Compatibilità

FREEFILE non è disponibile né in Turbo Basic né in BASICA.

GET

L'istruzione GET # legge un record di dati da un file ad accesso random o un numero specifico di caratteri da un file che è aperto nel modo BINARY.

```
GET #num, rec, variab  ' Legge un record da un file ad accesso random.
                       ' L'argomento rec è opzionale.

GET #num, rec          ' Legge un record nei campi definiti con FIELD.
                       ' L'argomento rec è opzionale.

GET #num, byte, variab ' Legge uno o più caratteri da un file che
                       ' è aperto nel modo BINARY.
```

L'argomento *#num* è il numero assegnato al file da un'istruzione OPEN precedente. (Il carattere # è facoltativo.) Per un file ad accesso random (aperto nel modo RANDOM), l'argomento *rec* è un intero che va da 1 fino al numero di record memorizzati nel file. GET # si pone direttamente su questo record e lo legge dal file. In alternativa, se si omette l'argomento *rec*, GET # legge il record *successivo* rispetto a quello a cui precedentemente si aveva avuto accesso tramite un'istruzione GET # o PUT #, oppure partendo dal record che era stato individuato da una precedente istruzione SEEK #. (È richiesta una virgola come indicatore di posizione, se si omette l'argomento *rec*.) L'argomento *variab* è solitamente il nome di una variabile record che corrisponde alla struttura record del file ad accesso random. QuickBASIC supporta anche lo stile BASICA per la programmazione di file di dati in cui l'istruzione GET # legge i dati nelle variabili FIELD stabilite. In questo stile, non più raccomandato come efficiente approccio alla gestione dei file di dati

nel linguaggio QuickBASIC, l'argomento *variab* viene omesso dall'istruzione GET #. (Si possono avere ulteriori notizie, riguardo questo metodo ormai superato, consultando il Capitolo 12.) Per un file che è aperto nel modo BINARY, l'argomento *byte* (che prende il posto dell'argomento *rec*) è un intero che può andare da 1 al numero totale di caratteri o byte del file. La lettura inizia perciò dalla posizione indicata con *byte*. Se si tralascia l'argomento *byte*, GET # inizia leggendo dal byte subito dopo l'ultimo carattere a cui aveva avuto accesso dal file o dal carattere che era stato individuato da una precedente istruzione SEEK #. L'argomento *variab* è una variabile stringa la cui lunghezza corrente (o lunghezza definita, nel caso di una variabile a lunghezza fissa) determina il numero di byte che verrà letto dal file. GET # è utilizzabile anche per leggere da un file di comunicazione. Consultare la voce OPEN COM nel Capitolo 13 per ulteriori informazioni.

Esempi di GET

La seguente trattazione presenta due esempi GET #. Il primo, preso dal programma *Agenda* (listato nel Capitolo 30), illustra l'utilizzo dell'istruzione GET # nella gestione di file ad accesso random. Il secondo è un semplice esercizio progettato per dimostrare la tecnica di lettura di un file binario.

Lettura di un file ad accesso random

Per gestire un file ad accesso random, un programma ha bisogno di un metodo per collocare record nel file. Il programma *Agenda* mantiene un database calendario, memorizzato in un file ad accesso random chiamato AGENDA.DAT, per memorizzare gli appuntamenti giornalieri dell'utente. Ogni record in questo database può contenere fino a dodici linee di testo che descrivono gli appuntamenti di una determinata data. L'utente può immettere record nel database in qualsiasi ordine. Per localizzare i record nel database, il programma mantiene anche un indice, memorizzato su disco in un file sequenziale chiamato AGENDA.NDX. Ogni linea del file indice è costituita da due dati, in questo formato:

data, posizione

Il primo valore è un intero lungo calcolato per rappresentare un data nel database calendario. Il secondo valore è un intero che rappresenta la posizione di quel record nel database AGENDA.DAT. Il programma *Agenda* tiene il file indice ordinato numericamente dagli interi *data*. (Il programma ha una routine di ordinamento con metodo Shell chiamata *OrdIndData*, che riorganizza l'indice quando nel database viene immesso un nuovo record.) Quando l'utente richiede una particolare data, una funzione di ricerca binaria chiamata *CercData* trova la data nel file indice e ritorna il numero intero che rappresenta la posizione del record corrispondente nel database. La seguente istruzione GET # (da una procedura chiamata *LeggiInData*) legge un record di data dal database aperto:

```
GET #1, numRec%, giorFiss
```

Il valore della variabile intera *numRec%* è la posizione del record della data che l'utente ha richiesto; questo è il valore che *CercData* ha trovato nel file indice. La variabile *giorFiss* è una variabile record in cui l'istruzione GET # legge gli appuntamenti del giorno.

Lettura di un file nel modo BINARY

Si può aprire in modo BINARY qualsiasi file. Solitamente lo scopo di far ciò è consentire ad un programma di accedere ai singoli byte del file piuttosto che ad unità più grandi di dati. Il seguente programma crea un file di testo e poi apre il file nel modo BINARY. Per descrivere l'utilizzo dell'istruzione GET # in questo modo, il programma legge il file due volte, ogni volta selezionando e visualizzando una singola linea di byte dal file. Per la prima lettura, il programma visualizza la linea come testo, per la seconda visualizza il codice ASCII equivalente di ogni byte che è letto dal file:

```
' Dimostrazione di lettura di file in modo binario.

OPEN "QUARTER.DAT" FOR OUTPUT AS #1

READ format$
READ recs%

' Crea un file sequenziale.
```

```
FOR i% = 1 TO recs%
   READ me$, an1%, an2%, an3%
   PRINT #1, USING format$; me$; an1%; an2%; an3%
NEXT i%
CLOSE #1

' Apre il file nel modo BINARY.

CLS
car$ = " "
OPEN "quarter.dat" FOR BINARY AS #1

PRINT
PRINT "Testo:"
PRINT

' Legge una linea dal file in modo binario
' e visualizza il testo sullo schermo.

FOR i% = LEN(format$) + 3 TO (LEN(format$) * 2) + 4
   GET #1, i%, car$
   PRINT car$;
NEXT i%

' Legge ancora la stessa linea e visualizza il
' codice ASCII di ogni carattere.

PRINT
PRINT "Codici ASCII:"
PRINT

SEEK #1, LEN(format$) + 3          ' Salta la prima linea.
FOR i% = 1 TO LEN(format$) + 2     ' Legge la seconda linea.
   GET #1, , car$
   PRINT USING "####"; ASC(car$);
   IF i% MOD 8 = 0 THEN PRINT
NEXT i%

CLOSE
END

DATA "\ \ $##,### $##,### $##,###"
DATA 3
DATA Gen, 19.000.000, 22.000.000, 20.000.000
DATA Feb, 12.500.000, 19.600.000, 17.200.000
DATA Mar, 18.200.000, 16.500.000, 15.400.000
```

Nella prima lettura, il contatore del ciclo FOR (*i%*) viene usato per selezionare ogni nuova posizione nel file; ogni carattere è letto dal file e poi visualizzato sullo schermo:

```
GET #1, i%, car$
PRINT car$;
```

Il programma ha precedentemente assegnato il valore di uno spazio alla variabile stringa di lunghezza variabile *car$*:

```
car$ = " "
```

Come risultato, ogni esecuzione dell'istruzione GET # legge un carattere dal file. Prima della seconda lettura, un'istruzione SEEK # seleziona una nuova posizione nel file, all'inizio della seconda linea del testo. Poi un ciclo FOR esegue l'istruzione GET # ripetutamente. Questa volta, però, l'argomento *byte* viene omesso; ogni GET # legge semplicemente il carattere successivo nel file:

```
SEEK #1 LEN(format$) + 3          ' Salta la prima linea.
FOR i% = 1 TO LEN(format$) + 2    ' Legge la seconda linea.
   GET #1, , car$
   PRINT USING "####"; ASC(car$);
NEXT i%
```

Si noti che il programma usa la funzione ASC per trovare l'equivalente ASCII di ogni carattere che viene letto; questo valore numerico del codice è poi visualizzato sullo schermo. Ecco l'output del programma:

```
Testo:

Feb L.12.500.000 L.19.600.000 L.17.200.000

Codici ASCII:

   70 101 98 32 32 36 49 50
   44  53 48 48 32 32 36 49
   57  44 54 48 48 32 32 36
   49  55 44 50 48 48 13 10
```

Compatibilità

Né Turbo Basic né BASICA supportano variabili record; entrambe le versioni del linguaggio perciò usano l'istruzione FIELD per stabilire la

struttura del record di un file ad accesso random. La sintassi dell'istruzione GET # in Turbo Basic e in BASICA è:

```
GET #num, rec
```

dove *#num* è il numero del file e *rec* è la posizione nel file del record in questione. GET # legge il record nelle variabili FIELD predefinite. Per leggere i byte di dati da un file che è aperto nel modo BINARY, Turbo Basic usa la funzione GET$ al posto di GET#:

```
GET$ #num, lungh, variab
```

Questa funzione legge un numero di byte uguali al valore dell'argomento intero *lungh*. La funzione inizia leggendo dalla posizione corrente nel file e memorizza i byte nella variabile assegnata come terzo argomento. Si può utilizzare l'istruzione SEEK # per selezionare una posizione nel file prima di GET$. Il primo byte in un file BINARY Turbo Basic ha il numero 0, non 1 come in QuickBASIC. BASICA non offre il modo BINARY per aprire i file.

INPUT

L'istruzione INPUT # legge i dati da un file sequenziale.

```
INPUT #num, variab, ...      ' È ammessa una lista di più variabili.
```

L'argomento *#num* è il numero assegnato al file da una precedente istruzione OPEN. La variabile o le variabili che seguono *#num* possono appartenere ad una stringa o ad un tipo numerico, ma devono corrispondere al tipo di dati che viene in effetti letto dal file. Ogni successiva istruzione INPUT # legge uno o più valori sequenzialmente dal file e memorizza i valori nelle variabili. Infatti, INPUT # sposta il puntatore in avanti nel file dopo aver letto un dato valore, così che il successivo INPUT # leggerà il valore seguente. Quando un'istruzione INPUT # legge l'ultimo dato nel file, la condizione di fine file diventa vera. Un programma può usare la funzione EOF per controllare questa condizione. Un tentativo di leggere oltre la fine del file finisce in un errore run-time ("Input Past End"). INPUT # riconosce uno dei seguenti caratteri ASCII come separatore tra due valori nel file:

- una virgola,
- un ritorno di carrello (ASCII 13),
- la combinazione ritorno carrello e salto riga (CR-LF) (ASCII 13 e 10),
- uno o più spazi (solo tra due numeri).

Un carattere di salto riga da solo non è riconosciuto come un delimitatore. Se un valore stringa contiene caratteri che normalmente sarebbero letti come delimitatori, la stringa deve essere chiusa tra virgolette all'interno del file. WRITE # è analoga a INPUT #. L'istruzione WRITE # scrive i dati in un file sequenziale, fornendo automaticamente i delimitatori che INPUT # riconosce: virgole tra i dati, e virgolette che delimitano le stringhe. (Si può anche usare l'istruzione INPUT # per leggere un file cui si siano forniti esplicitamente i delimitatori necessari nell'istruzione PRINT #.) Sia WRITE # sia PRINT # inviano una combinazione di ritorno carrello e salto riga alla fine di una linea.

Un esempio di INPUT

Il programma *Agenda* (presentato nel Capitolo 30) apre l'indice chiamato AGENDA.NDX come file di dati sequenziale. Ecco come viene aperto il file per la lettura:

```
OPEN "AGENDA.NDX" FOR INPUT AS #2
```

Il file contiene coppie di interi in una lista che il programma calcola in modo che sia lunga *contImm%* linee. Il primo numero in ogni coppia è un intero lungo che rappresenta una data ed il secondo intero indica la posizione del record di quella data nel database ad accesso random che il programma gestisce. Ecco come potrebbero presentarsi alcune linee di questo file:

```
32676.32
32677.29
32678.27
32679.24
32680.34
```

Il programma legge l'indice intero in un vettore di record, *indData*, che ha i campi *dataLung* e *posData*.

```
FOR imm% = 1 TO contImm%
   INPUT #2 indData(imm%).dataLung, indData(imm%).posData
NEXT imm%
```

Come si può vedere, i nomi delle variabili nella lista INPUT # possono apparire come elementi del vettore o campi del record.

Compatibilità

L'istruzione INPUT # è la stessa sia in Turbo Basic sia in BASICA.

INPUT$

La funzione INPUT$ legge caratteri da un file o dalla tastiera e ritorna i dati come un valore stringa.

```
INPUT$(byte, #num)    ' Legge byte caratteri dal file identificato
                      ' come #num (il simbolo # è opzionale).

INPUT$(byte)          ' Legge byte caratteri dalla tastiera.
```

L'argomento *byte* è un intero che specifica il numero di caratteri che INPUT dovrebbe leggere. Il secondo argomento opzionale, *#num*, è il numero di file assegnato ad un file da una precedente istruzione OPEN. Se si inserisce il secondo argomento, INPUT$ legge dal file specificato iniziando dalla posizione corrente nel file. Senza l'argomento *#num*, INPUT$ legge i dati dalla tastiera. Come strumento d'input, la funzione INPUT$ è molto diversa dall'istruzione INPUT: INPUT$ non visualizza i caratteri immessi sullo schermo come l'utente li digita. Inoltre, INPUT$ blocca la lettura non appena un numero di caratteri pari a *byte* è stato ricevuto; l'utente non deve premere il tasto Invio per completare l'input.

Un esempio di INPUT$

Il seguente esercizio usa INPUT$ per leggere tre linee da un file di testo:

```
contLin% = 3
nomeFile$ = "PROVA.DAT"

OPEN nomeFile$ FOR INPUT AS #1
dimLin% = LOF(1) / contLin%

CLS
FOR linCorr% = 1 TO contLin%
   PRINT INPUT$(dimLin% - 2, 1)
   cr$ = INPUT$(2, 1)
NEXT linCorr%

CLOSE #1
```

Queste linee vengono lette una alla volta dall'interno di un ciclo FOR. Si noti che vi sono due chiamate alla funzione INPUT$. La prima chiamata legge una linea intera ad eccezione dei due caratteri finali della linea, il ritorno carrello e il fine linea (ASCII 13 e 10). Un'istruzione PRINT visualizza la linea sullo schermo. Poi la seconda chiamata ad INPUT$ legge e scarta i due caratteri finali della linea.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione INPUT$.

LINE INPUT

L'istruzione LINE INPUT # legge una linea di testo da un file sequenziale aperto.

```
LINE INPUT #num, variab   ' Deve essere una variabile stringa.
```

L'argomento *#num* è il numero assegnato al file da un'istruzione OPEN precedente. Il secondo argomento è il nome di una variabile stringa, a lunghezza fissa o variabile. L'istruzione LINE INPUT # legge una linea intera di testo dal file e memorizza la linea nella variabile stringa, anche se il testo contiene virgole o altri caratteri che normalmente potrebbero servire come delimitatori tra valori di dati. Una combinazione ritorno carrello/fine linea segna la fine di una linea; LINE INPUT # non memorizza questi due

caratteri nella variabile stringa. Ogni successiva istruzione LINE INPUT # legge la linea successiva dal file, sequenzialmente dall'inizio alla fine. Quando un'istruzione LINE INPUT # legge l'ultima linea nel file, la condizione di fine file diventa vera; un programma può usare la funzione EOF per controllare questa condizione. Un tentativo di leggere oltre la fine del file finisce in un errore run-time ("Input past end").

Un esempio di LINE INPUT

Un programma chiamato *Linee*, listato e discusso nella voce EOF, è progettato per aprire un file nominato e contare le linee non vuote. Il programma utilizza l'istruzione LINE INPUT # per leggere ogni linea:

```
contLin% = 0
DO WHILE NOT EOF(1)
   LINE INPUT #1, inLin$
   IF LTRIM$(inLin$) <> "" THEN contLin% + 1
LOOP
```

Un ciclo DO controlla la lettura linea per linea del file. Dopo che ogni linea è stata letta, il programma esamina il valore di *inLin$*. Se la stringa è composta da caratteri oltre agli spazi, il programma incrementa il contatore *contLin%* di 1. Il processo continua fin che la funzione EOF non ritorna un valore vero. (Vedere la funzione EOF per ulteriori dettagli.)

Compatibilità

LINE INPUT # è uguale sia in Turbo Basic sia in BASICA.

LOC

La funzione LOC ritorna un intero che rappresenta la posizione corrente in un file aperto.

```
LOC(num)
```

L'argomento *num* è il numero assegnato al file da una precedente istruzione OPEN. Il valore di ritorno dipende dal modo di accesso:

- Per un file ad accesso random, LOC ritorna il numero dell'ultimo record in cui si è entrati.
- Per un file ad accesso sequenziale, LOC ritorna un intero che rappresenta il numero del "record" corrente a 128 byte.
- Per un file aperto nel modo binario, LOC ritorna il numero dell'ultimo byte a cui si ha avuto accesso.

LOC può anche essere usato con un file di comunicazione, per determinare il numero di caratteri d'input disponibili.

Un esempio LOC

LOC è una delle due funzioni disponibili in QuickBASIC per indicare la posizione corrente in un file. L'altra è SEEK. Il seguente esercizio illustra la differenza tra LOC e SEEK:

```
' Dimostrazione SEEK e LOC.

OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = 600
DIM inVuot AS STRING * 600
GET #1, 10, inVuot

OPEN "AGENDA.NDX" FOR INPUT AS #2
FOR i% = 1 TO 20
   LINE INPUT #2, inLin$
NEXT i%

OPEN "PROVA.DAT" FOR BINARY AS #3
inCar$ = " "
GET #3, 25, inCar$

CLS
PRINT , "SEEK()", "LOC()"
PRINT "RANDOM", SEEK(1), LOC(1)
PRINT "INPUT", SEEK(2), LOC(2)
PRINT "BINARY", SEEK(3), LOC(3)

RESET
```

Il programma apre tre file in diversi modi e legge dati da ogni file. Poi una sequenza di istruzioni PRINT visualizza i valori correnti SEEK e LOC per ogni file. Ecco l'output del programma:

```
         SEEK()       LOC()
RANDOM    11           10
INPUT     192          2
BINARY    26           25
```

Per i file binari e ad accesso random, LOC ritorna il numero dell'*ultima* voce a cui si ha avuto accesso, mentre SEEK dà la posizione della voce *successiva*. Per file sequenziali, LOC dà un intero che rappresenta il record corrente a 128 byte; SEEK identifica la posizione del byte successivo.

Compatibilità

Turbo Basic e BASICA hanno la funzione LOC, ma non la funzione SEEK.

LOCK...UNLOCK

L'istruzione LOCK blocca uno o più record in un file specificato, impedendo l'accesso ad altri utenti in un ambiente di rete. L'istruzione UNLOCK corrispondente sblocca i record.

```
LOCK #num                    ' Blocca tutti i record.
LOCK #num, rec               ' Blocca un record.
LOCK #num, rec1 TO rec2      ' Blocca un intervallo di record.
LOCK #num, TO rec            ' Blocca l'intervallo, iniziando da 1.

UNLOCK #num                  ' Sblocca tutti i record.
UNLOCK #num, rec             ' Sblocca un record.
UNLOCK #num, rec1 TO rec2    ' Sblocca un intervallo di record.
UNLOCK #num, TO rec          ' Sblocca l'intervallo, da 1.
```

L'argomento *#num* è il numero di file assegnato al file da una precedente istruzione OPEN. Gli argomenti *rec* rappresentano i numeri di record in un file ad accesso random o posizioni di byte in un file binario. Si può bloccare un singolo record, un intervallo di record da una posizione all'altra *(rec1 TO*

rec2), oppure un intervallo di record partendo dal primo nel file *(TO rec)*. L'argomento record non ha effetto in un file ad accesso sequenziale; un'istruzione LOCK applicata ad un file sequenziale blocca sempre l'intero file. Ogni istruzione LOCK deve essere seguita nel programma da una corrispondente istruzione UNLOCK con esattamente gli stessi valori d'argomento. La rete è supportata solo nel DOS 3.1 o successivi. Si noti che le clausole blocca e ACCESS nell'istruzione OPEN riguardano l'uso di un file in una rete.

Compatibilità

Turbo Basic e BASICA non supportano l'istruzione LOCK.

LOF

La funzione LOF ritorna un intero che rappresenta le dimensioni (o la "lunghezza") di un file del disco.

```
LOF(num)
```

L'argomento *num* è il numero del file assegnato al file da una precedente istruzione OPEN. Il valore di ritorno è il numero di byte di dati memorizzati nel file. Si può usare questa funzione su un file aperto del disco, sequenziale, ad accesso random o binario.

Un esempio di LOF

LOF è particolarmente utile per determinare il numero di record memorizzati in un file ad accesso random. Questa espressione dà il conteggio dei record:

```
LOF(num) / lunghRec
```

dove *lunghRec* è la lunghezza in byte della struttura record del file. È interessante sottolineare che si può usare la funzione LEN in QuickBASIC

per determinare la lunghezza totale in byte di una variabile record. LOF e LEN insieme forniscono un modo semplice per trovare il numero di record in un file. Ad esempio, il programma *Agenda* (presentato nel Capitolo 30) lavora con il file AGENDA.DAT ad accesso random. Una struttura record associata chiamata *recGior* viene definita in un'istruzione TYPE vicino all'inizio del programma. Il programma dichiara molte variabili record che appartengono a questo tipo, secondo le esigenze; ad esempio, la seguente istruzione DIM (presa da una procedura chiamata *AprDatab)* dichiara la variabile record *giorno*:

```
DIM giorno AS recGior
```

Il programma usa questa particolare variabile nel processo di apertura del file e determina il conteggio attuale:

```
OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = LEN(giorno)
contImm% = LOF(1) / LEN(giorno)
```

La funzione LOF fornisce le dimensioni del file in byte e dividendole per la lunghezza del record *giorno* dà il conteggio record corrente. Il programma memorizza questo conteggio nella variabile globale *contImm%*.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione LOF, ma non supportano le variabili record. Per usare LOF per trovare il numero di record in un file ad accesso random, bisogna conoscere in anticipo la lunghezza della struttura record.

OPEN

L'istruzione OPEN apre un file del disco in uno o più modi o attiva un dispositivo per operazioni d'input ed output.

Ambiente monoutente:

```
OPEN nome FOR modo #num      ' Modo binario o sequenziale.
```

```
OPEN nome FOR RANDOM AS #num LEN=lungh     ' Accesso random.
```

Ambiente pluriutente:

```
OPEN nome FOR modo ACCESS modAcc modBloc As #num LEN=lungh
```

L'argomento *nome* è un valore stringa che dà il nome del file da aprire. Si può esprimere questa stringa come un valore literal, una variabile o un'espressione. Le caratteristiche del nome di un file sono stabilite dal DOS. La base del nome può contenere fino a otto caratteri, seguiti da un'estensione opzionale a tre caratteri. Il nome e l'estensione sono separati da un punto. Si può inserire anche il nome di un drive e/o di un percorso, per identificare la posizione del file. (In alternativa, la stringa *nome* può presentarsi come il nome di un dispositivo, KYBD:, SCRN:, COM*n*:, LPT*n*: o CONS:, in questo caso, OPEN attiva l'input o l'output per il dispositivo selezionato.) La clausola AS assegna un numero al file. Questo valore, un intero da 1 a 255, deve essere unico per ogni file che viene aperto in un determinato momento. Le istruzioni seguenti che lavorano con il file si riferiscono ad esso attraverso questo numero. Quando si chiude un file, il suo numero è di nuovo disponibile per un altro file.

Modi per aprire un file

La clausola FOR specifica il modo con cui viene aperto il file. Esistono cinque possibilità: RANDOM apre un file ad accesso random; INPUT, OUTPUT e APPEND rappresentano i modi ad accesso sequenziale; e BINARY apre un file nel modo binario. Un file dato può essere aperto contemporaneamente in tre diversi modi: RANDOM, BINARY e INPUT. Comunque, un file che è aperto tramite OUTPUT o APPEND deve essere chiuso prima che possa essere riaperto in un diverso modo.

File ad accesso random

Se si omette la clausola FOR, il modo di default è RANDOM. Un file ad accesso random ha una struttura record a lunghezza fissa, che consente a

QuickBASIC di accedere ai record direttamente usando il loro numero, secondo qualsiasi ordine. Una volta che un file ad accesso random è aperto, un programma può leggervi dei record, aggiungerne o scrivere record modificati nelle posizioni dei record esistenti. L'istruzione TYPE è in genere usata per definire una struttura per un file ad accesso random. Poi un programma dichiara una o più variabili record per memorizzare unità di dati lette dal file o scritte in esso. (Vedere il Capitolo 3 per informazioni sulle variabili record.) La clausola LEN nell'istruzione OPEN specifica la lunghezza del record nella struttura di un file ad accesso random. Il modo più semplice per dare questa lunghezza è dichiarare una variabile record che corrisponda alla struttura record del file e poi usare la funzione LEN incorporata di QuickBASIC per calcolare la lunghezza del record:

```
LEN = LEN(varRec)
```

(Vedere gli esempi che seguono per ulteriori informazioni.) La lunghezza massima consentita ad un record in QuickBASIC è di 32.767 byte. I valori numerici sono memorizzati in un file ad accesso random negli appropriati formati non ASCII. (I formati standard IEEE vengono usati in QuickBASIC 4.0 e 4.5.) Questi formati non sono leggibili come testo. Comunque, non è necessaria nessuna conversione del formato in QuickBASIC 4.5, purché si definisca una variabile record come mezzo per leggere e scrivere dati. (Nel vecchio stile di gestione dei file ad accesso random, i valori numerici scritti su un file o letti da esso dovevano essere convertiti tra diversi formati di memorizzazione. Vedere il Capitolo 12 per ulteriori dettagli.)

I file ad accesso sequenziale

L'istruzione OPEN apre un file per un'operazione sequenziale alla volta: la lettura da un file esistente (INPUT), la scrittura di informazioni in un file ricreato (OUTPUT) o l'aggiunta di informazioni alla fine di un file (APPEND). Ognuno di questi modi ha le proprie speciali caratteristiche:

- Un errore run- time si verifica se QuickBASIC non riesce a trovare il file che si cerca di aprire per INPUT; i programmi contengono spesso una gestione degli errori per manipolare questo potenziale errore.

- Se si apre un file tramite OUTPUT con un nome che è già stato usato, QuickBASIC scrive sopra al file esistente.
- Il modo APPEND prepara a scrivere nuovi dati alla fine di un file esistente; comunque, se il file nominato non esiste su disco, QuickBASIC lo crea. In questo caso, APPEND è uguale a OUTPUT.

Si può includere una clausola LEN opzionale in un'istruzione OPEN per un file ad accesso sequenziale; se c'è, la clausola specifica le dimensioni del buffer d'input per leggere o scrivere dati. Il buffer di default è di 512 byte.

Il modo BINARY

Alcuni file di dati possono essere aperti nel modo BINARY. Questo modo dà l'accesso diretto ai singoli byte di dati nel file. L'istruzione GET # legge ogni carattere da una specifica posizione nel file e l'istruzione PUT # scrive i caratteri nel file. La clausola LEN dell'istruzione OPEN non viene usata nel modo BINARY.

I file in un ambiente multiutente

L'istruzione OPEN ha due clausole opzionali da usare in un ambiente della rete. Queste clausole richiedono il DOS 3.0 o programmi successivi. La clausola ACCESS specifica il tipo di attività che verrà eseguita con il file. Tre opzioni di accesso sono disponibili per il *modAcc:* READ, WRITE o READ WRITE. La clausola *modBloc* indica le operazioni possibili per altri utenti. Le opzioni *modBloc* sono SHARED (che consente lettura e scrittura); LOCK READ (la lettura non è consentita); LOCK WRITE (la scrittura non è consentita) e LOCK READ WRITE (non sono consentite né la lettura né la scrittura).

Sintassi alternativa per il comando OPEN

QuickBASIC supporta anche una sintassi vecchio stile per l'istruzione OPEN. Mentre non è raccomandabile usarla nei programmi QuickBASIC,

questa sintassi compare talvolta nei programmi scritti in versioni precedenti del linguaggio:

```
OPEN modo, #num, nome, lungh
```

L'argomento *modo* è un valore stringa che consiste in una sola lettera: I, O, A, R o B, che rappresentano i modi INPUT, OUTPUT, APPEND, RANDOM e BINARY. Il secondo argomento, *#num*, dà il numero del file ed il terzo argomento è un valore stringa che indica il nome del file. L'argomento finale è un intero che rappresenta la lunghezza del record in un file ad accesso random. La sintassi non supporta le clausole multiutente.

Esempi di OPEN

Il programma *Agenda* (presentato nel Capitolo 30) lavora con un file sequenziale, AGENDA.NDX, e uno ad accesso random, AGENDA.DAT. Il file sequenziale, che serve da indice per i record memorizzati nel file ad accesso random, viene aperto due volte durante l'esecuzione di un programma, una volta perché il programma possa leggere l'indice nella memoria e un'altra perché possa memorizzare su disco una nuova versione dell'indice. La procedura che apre il file per la lettura *(AprIndDat)* ha la precauzione di attivare la gestione degli errori, anticipando la possibilità che il file non venga trovato sul disco:

```
ON ERROR GOTO noIndFile
   OPEN "AGENDA.NDX" FOR INPUT AS #2
ON ERROR GOTO 0
```

Supponendo che l'istruzione OPEN venga eseguita con successo, il programma continua a leggere l'indice in un vettore e poi richiude il file. Questa precauzione non è necessaria per scrivere l'indice sul disco alla fine dell'esecuzione del programma. Ciò viene eseguito in una procedura che si chiama *IndCorr*. In effetti, il programma crea un nuovo file AGENDA.NDX, indipendentemente dal fatto che esista o meno una versione precedente del file:

```
OPEN "AGENDA.NDX" FOR OUTPUT AS #2
```

Il database stesso, AGENDA.DAT, è aperto all'inizio dell'esecuzione e rimane aperto fino alla fine. Il programma dichiara una variabile record, *giorno*, per stabilire la lunghezza del record del file. L'apertura si ha in una procedura chiamata *AprDatab:*

```
DIM giorno AS recGior
OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = LEN(giorno)
```

Si noti l'uso della funzione LEN per dare la lunghezza della variabile record.

Compatibilità

Per un ambiente monoutente le istruzioni OPEN di Turbo Basic e BASICA hanno la stessa sintassi di quelle del QuickBASIC. Comunque, né Turbo Basic né BASICA supportano variabili record. Per questo motivo, un'istruzione OPEN per un file ad accesso random è quasi sempre preceduta da un'istruzione FIELD, che stabilisce la struttura record per il file. Le opzioni multiutente non sono supportate né in Turbo Basic né in BASICA.

PRINT # E PRINT # USING

L'istruzione PRINT # memorizza informazioni in un file di dati sequenziale.

```
PRINT #num,                   ' Memorizza una linea vuota nel file.

PRINT #num, val; ...; val;    ' Memorizza i valori fianco a fianco.
                              ' Il punto e virgola finale è opzionale.

PRINT #num, val, ..., val,    ' Memorizza i valori nelle zone a 14 colonne.
                              ' La virgola finale è facoltativa.

PRINT #num, USING form; val; ...; val;     ' Memorizza dati formattati.
```

L'argomento *#num* è il numero assegnato al file da un'istruzione OPEN precedente, e apre il file nel modo OUTPUT o APPEND. PRINT # e PRINT # USING hanno tutte le stesse opzioni come le analoghe istruzioni output dello schermo, PRINT e PRINT USING. La lista di dati nelle istruzioni PRINT# può comprendere valori stringa o numerici, espressi come costanti,

variabili o espressioni. Un'istruzione PRINT # che non contiene nessuna lista di dati dà come risultato una linea vuota. La punteggiatura nell'istruzione PRINT # determina gli spazi tra i valori di output. Il punto e virgola colloca i valori fianco a fianco, le virgole fanno sì che QuickBASIC memorizzi i valori in *zone* a 14 colonne. Un punto e virgola o una virgola collocate alla fine di un'istruzione PRINT # o PRINT # USING elimina l'output dei caratteri ritorno carrello e salto riga, che vengono normalmente inviati alla fine di ogni linea. PRINT # USING accetta una *stringa di formato*, costituita da speciali caratteri o da caratteri che descrivono l'aspetto desiderato dei dati emessi. (Si tenga presente che nella sintassi di questa istruzione è richiesta una virgola tra l'argomento *#num* e la clausola USING.) Per ulteriori dettagli riguardanti queste opzioni, vedere le voci PRINT e PRINT USING. Le istruzioni PRINT # e PRINT USING # vengono spesso usate per creare file di testo facili da leggere per gli *utenti*. D'altra parte, se si vuol creare un file di dati sequenziale che possa esser letto convenientemente da un *programma*, in particolare con l'istruzione INPUT#, bisognerebbe nella maggior parte dei casi usare l'istruzione WRITE #, invece di PRINT #, per memorizzare i dati nel file. Diversamente da PRINT#, l'istruzione WRITE # invia appropriati delimitatori (come virgole e virgolette) al file, così che l'istruzione INPUT# può distinguere facilmente un valore da un altro.

Un esempio di PRINT

Un programma dimostrativo presentato nella voce GET# comprende un frammento che costruisce un file di testo dai valori memorizzati in una serie di istruzioni DATA. Il file si chiama QUARTER.DAT:

```
OPEN "PROVA.DAT" FOR OUTPUT AS #1
```

Il seguente ciclo FOR scrive i dati nel file:

```
FOR i% = 1 TO recs%
   READ me$, an1%, an2%, an3%
   PRINT #1, USING format$; me$; an1%; an2%; an3%
NEXT i%
```

Si noti che il programma usa un'istruzione PRINT # USING per inviare singole linee di testo al file. La prima delle istruzioni DATA che il programma legge contiene la stringa di formato per l'istruzione PRINT # USING; le successive linee DATA contengono i dati per il file:

```
DATA "\ \ L.##.###.###"
DATA 3
DATA Gen, 19.000.000, 22.000.000, 20.000.000
DATA Feb, 12.500.000, 19.600.000, 17.200.000
DATA Mar, 18.200.000, 16.500.000, 15.400.000
```

Quando il programma è completo, ecco come si presenta il file:

```
Gen L. 19.000.000 L. 22.000.000 L. 20.000.000
Feb L. 12.500.000 L. 19.600.000 L. 17.200.000
Mar L. 18.200.000 L. 16.500.000 L. 15.400.000
```

Compatibilità

Le istruzioni PRINT# e PRINT # USING sono uguali sia in Turbo Basic sia in BASICA.

PUT

L'istruzione PUT # scrive un record di dati in un file ad accesso random o un numero specifico di caratteri in un file che è aperto nel modo BINARY.

```
PUT #num, rec, variab     ' Scrive un record in un file ad accesso random.
                          ' L'argomento rec è opzionale.

PUT #num, rec,            ' Scrive un record nelle variabili FIELD.
                          ' L'argomento rec è opzionale.

PUT #num, byte, variab    ' Scrive uno o più caratteri in un file
                          ' che è aperto nel modo BINARY.
```

L'argomento *#num* è il numero assegnato al file da un'istruzione OPEN precedente. (Il carattere # è facoltativo.) Per un file ad accesso random (aperto nel modo RANDOM), l'argomento *rec* è un intero che generalmente

ha un intervallo da 1 fino al conteggio record più 1. PUT # si muove direttamente verso la posizione di questo record e scrive un record nel file. Se la posizione è quella di un record esistente, il record precedente viene completamente sovrascritto dal nuovo record. Comunque, se il valore della posizione del record è maggiore di 1 rispetto al numero corrente di record del file, PUT # aggiunge un nuovo record. Se si omette l'argomento *rec*, PUT # scrive alla posizione del record *successivo* rispetto a quello a cui precedentemente aveva avuto accesso tramite un'istruzione GET # o PUT #, oppure partendo dalla posizione del record che era stata selezionata da un precedente comando SEEK #. (È comunque richiesta una virgola come indicatore di posizione nella sintassi di PUT #, se si omette l'argomento *rec*.) L'argomento *variab* è solitamente il nome di una variabile record che corrisponde alla struttura record del file ad accesso random. QuickBASIC supporta anche lo stile BASICA di gestione di file di dati in cui l'istruzione PUT # scrive i dati nel file da variabili FIELD stabilite. In questo stile, non più raccomandato come efficiente approccio alla programmazione dei file di dati nel linguaggio QuickBASIC, l'argomento *variab* viene omesso dall'istruzione PUT #. (Si possono avere ulteriori notizie riguardo questo metodo ormai superato, consultando il Capitolo 12.) Per un file che è stato aperto nel modo BINARY, l'argomento *byte* (che prende il posto dell'argomento *rec*) è un intero che generalmente può andare da 1 al conteggio corrente di byte del file più 1. La scrittura inizia da questo byte nel file. Se si tralascia l'argomento *byte*, PUT # inizia scrivendo dal byte dopo l'ultimo carattere a cui aveva avuto accesso dal file, o dal carattere che era stato selezionato da una precedente istruzione SEEK #. L'argomento *variab* è una variabile stringa la cui lunghezza determina il numero di byte che verrà scritto nel file. PUT # può essere usato anche per inviare dati ad un file di comunicazione. Consultare la voce OPEN COM nel Capitolo 13 per ulteriori informazioni.

Un esempio PUT

Il programma *Agenda*, presentato nel Capitolo 30, usa l'istruzione PUT # per riesaminare un record esistente del database agenda o per aggiungere un nuovo record alla fine del database. Ecco la sequenza di eventi che porta a questa istruzione nel programma:

1. L'utente richiede uno specifico record del calendario ed il programma lo cerca attraverso la data. Se il record esiste già, il programma lo legge e ne visualizza il contenuto sullo schermo; altrimenti, appare lo schermo vuoto.
2. Nel dialogo d'input che segue, l'utente immette nuove informazioni nel record. Quando ha finito, il programma chiede la conferma per salvare il record.
3. Se il record non esiste, il programma scrive le nuove informazioni sul record precedente; altrimenti, se è un record nuovo, lo aggiunge alla fine del file del database.

Il programma memorizza gli appuntamenti immessi nella data scelta come campi di una variabile record chiamata *giorno*. La variabile *numRec%* contiene la posizione del record nel database, o il numero di un record esistente, o un intero che è più grande di 1 rispetto al conteggio di record corrente del file. Dati questi due valori, la seguente istruzione scrive il record nel file del database:

```
PUT #1, numRec%, giorno
```

(Questa istruzione si trova in una procedura che si chiama *SalvGior*.)

Compatibilità

Né Turbo Basic né BASICA supportano variabili record; entrambe le versioni del linguaggio perciò usano l'istruzione FIELD per stabilire la struttura record di un file ad accesso random. La sintassi dell'istruzione PUT # nel Turbo Basic e in BASICA è:

```
PUT #num, rec
```

dove *#num* è il numero del file e *rec* è la posizione in cui verrà scritto il record. Il record scritto nel file è rappresentato dai campi di FIELD. Per scrivere i byte di dati in un file che è aperto nel modo BINARY, Turbo Basic usa la funzione PUT$:

```
PUT$ #num, variab
```

Questa funzione scrive un numero di byte uguale alla lunghezza dell'argomento della variabile. La funzione inizia a scrivere nella posizione corrente nel file. Si può usare l'istruzione SEEK# per scegliere una posizione nel file prima di PUT$; il primo byte in un file BINARY del Turbo Basic ha come numero 0, non 1 come in QuickBASIC. BASICA non offre il modo BINARY per aprire i file.

RESET

L'istruzione RESET chiude tutti i file aperti.

```
RESET
```

Questa istruzione è un'alternativa a CLOSE. Per chiudere tutti i file aperti si puo usare o l'istruzione RESET o CLOSE senza argomenti. (Vedere la voce CLOSE per ulteriori dettagli.)

Compatibilità

Sia Turbo Basic sia BASICA supportano questa istruzione.

FUNZIONE SEEK

La funzione SEEK ritorna un intero che rappresenta la posizione successiva alla corrente in un file aperto.

```
SEEK(num)
```

L'argomento *num* è il numero di file assegnatogli da una precedente istruzione OPEN. Il valore di ritorno dipende dal modo di accesso al file:

- Per un file ad accesso random, SEEK ritorna il numero del record seguente a cui si avrà accesso se non viene selezionata esplicitamente un'altra posizione.

- Per i file binari e ad accesso sequenziale, SEEK ritorna un intero che rappresenta il byte successivo a cui si accederà se non viene esplicitamente scelta un'altra posizione.

Un esempio della funzione SEEK

SEEK è una delle due funzioni disponibili in QuickBASIC per modificare la posizione corrente in un file. L'altra è LOC. Un esercizio presentato alla voce LOC illustra la differenza tra SEEK e LOC. Il programma apre tre file in diversi modi e legge i dati da ogni file. Poi una sequenza di istruzioni PRINT visualizza i valori correnti di SEEK e LOC per ogni file:

```
PRINT , "SEEK()", "LOC()"
PRINT "RANDOM", SEEK(1), LOC(1)
PRINT "INPUT", SEEK(2), LOC(2)
PRINT "BINARY", SEEK(3), LOC(3)
```

Ecco l'output del programma:

```
            SEEK()       LOC()
RANDOM       11           10
INPUT        192          2
BINARY       26           25
```

Per tutti i file, SEEK ritorna il numero delle voci *successive* a cui è possibile aver accesso, mentre LOC dà la posizione dell'*ultima* voce. (Per i file sequenziali, LOC dà un intero che rappresenta il record corrente a 128 byte.)

Compatibilità

Né Turbo Basic né BASICA hanno la funzione SEEK.

ISTRUZIONE SEEK

L'istruzione SEEK seleziona la posizione di un record in un file ad accesso random o la posizione di un byte in un file aperto in alcuni altri modi.

```
SEEK #num, posiz
```

L'argomento *#num* è il numero di file assegnatogli da una precedente istruzione OPEN. L'argomento *posiz* è un intero che rappresenta la posizione che l'istruzione SEEK sceglie nel file. Una successiva istruzione di input o di output inizia leggendo o scrivendo in questa nuova posizione selezionata. Il primo record in un file ad accesso random ha come numero 1. L'istruzione SEEK sceglie una posizione record da 1 fino al numero di record nel file. In altri modi di accesso (INPUT, OUTPUT, APPEND e BINARY), il primo *byte* ha il numero 1 e l'istruzione SEEK seleziona una posizione del byte nel file. SEEK non è richiesta nella maggior parte dei programmi ad accesso random, perché le istruzioni GET # e PUT # hanno argomenti di posizione propri. In effetti, GET # e PUT # eseguono un'implicita operazione SEEK prima di leggere o scrivere nel file.

Un esempio dell'istruzione SEEK

Un programma presentato nella voce GET # contiene una descrizione dell'istruzione SEEK. Il programma apre un file di testo chiamato QUARTER.DAT nel modo BINARY:

```
OPEN "prova.dat" FOR BINARY AS #1
```

Per leggere la seconda linea del file, il programma usa un'istruzione SEEK per saltare un numero calcolato di caratteri nel file:

```
SEEK #1, LEN(format$) + 3
```

Vedere la voce GET # per ulteriori dettagli.

Compatibilità

Turbo Basic supporta l'istruzione SEEK solo per i file BINARY. L'istruzione non è disponibile in BASICA.

WRITE

L'istruzione WRITE # invia informazioni ad un file di dati sequenziale e fornisce appropriati delimitatori tra i dati.

```
WRITE #num,                  ' Scrive una linea vuota.

WRITE #num, val, ..., val    ' Invia i dati al file.
```

L'argomento *#num* è il numero di file assegantogli da una precedente istruzione OPEN. La lista di dati nell'istruzione WRITE può comprendere valori stringa o numerici, espressi come costanti, variabili o espressioni. Se nella lista si inserisce un'espressione, WRITE # la valuta ed invia il valore risultante al file. Un'istruzione WRITE # che non contiene nessuna lista di dati scrive una linea vuota. A differenza dell'istruzione PRINT #, WRITE # separa ogni output dal successivo con una virgola che viene inviata al file. In WRITE # i valori stringa sono tra virgolette. Per questo motivo, WRITE # è l'istruzione ideale per creare un file che successivamente verrà letto dal comando INPUT #.

Un esempio WRITE

Il programma *Agenda*, presentato nel Capitolo 30, crea un file sequenziale chiamato AGENDA.NDX. Questo file ha le funzioni di indice nel database del calendario che il programma gestisce. Il seguente frammento, preso da una procedura chiamata *IndCorr*, mostra come il programma scrive una lista di dati nel file:

```
OPEN "AGENDA.NDX" FOR OUTPUT AS #2

FOR imm% = 1 TO contImm%
   WRITE #2, indData(imm%).lunghData, indData(imm%).posData
NEXT imm%
```

In questo caso, i due valori interi che WRITE # invia al file sono due campi da un vettore di record. Il ciclo FOR organizza l'invio di una lista intera di questi valori al file. Ecco un esempio di questa lista:

```
32677,29
32678,27
32679,24
32680,34
32681,25
32683,38
32684,39
32691,22
32697,21
32703,37
```

Compatibilità

Sia Turbo Basic sia BASICA supportano l'istruzione WRITE #.

Capitolo 12

Input ed output su disco: stile BASICA

Il linguaggio interprete BASICA, ed alcune versioni precedenti del compilatore QuickBASIC, per l'esecuzione di operazioni d'input e output su un file ad accesso casuale richiedevano una sequenza di fasi piuttosto approssimativa, che coinvolgevano le seguenti istruzioni e funzioni:

- L'istruzione FIELD, per definire la struttura del record di un file ad accesso casuale.
- Le istruzioni LSET e RSET per copiare i dati nelle variabili FIELD prima di un'operazione di scrittura.
- Le funzioni MKI$, MKL$, MKS$ e MKD$ per creare formati di numeri a lunghezza fissa che rappresentano i dati numerici in un file.
- Le funzioni CVI, CVL, CVS e CVD per convertire quegli stessi formati in valori numerici.

Le versioni più recenti di QuickBASIC hanno semplificato molto questo processo introducendo nel linguaggio le strutture record definite dall'utente. Comunque, le istruzioni e funzioni stile BASICA sono ancora disponibili per fornire la compatibilità con le versioni precedenti del BASIC Microsoft. Questo capitolo descrive questi strumenti ora superati. Si può così confrontare il vecchio approccio con il nuovo: le voci di questo capitolo presentano

sia esempi dello stile BASICA sia dello stile QuickBASIC. In particolare si troveranno le seguenti illustrazioni sparse nel corso del capitolo:

- Le voci MKI$, MKL$, MKS$ e MKD$ presentano un programma che usa le tecniche BASICA per creare un file ad accesso casuale.
- Le voci LSET e RSET mostrano come lo stesso compito può essere eseguito usando le nuove tecniche di QuickBASIC.
- Le voci CVI, CVL, CVS e CVD danno un esempio dell'approccio BASICA per leggere dati da un file ad accesso casuale.
- La voce FIELD dà una versione QuickBASIC dello stesso programma.

CVI, CVL, CVS, CVD

CVI, CVL, CVS e CVD traducono in numeri i contenuti delle variabili FIELD ad accesso casuale. Dato un argomento del campo stringa che rappresenta un formato standard del numero, ognuna di queste funzioni di conversione ritorna l'equivalente valore di tipo numerico.

```
CVI(strInt)     ' Converte una stringa a due byte in un intero.
CVL(strlung)    ' Converte una stringa a quattro byte
                ' in un intero lungo.
CVS(strSing)    ' Converte una stringa a quattro byte
                ' in una in precisione semplice.
CVD(strDop)     ' Converte una stringa ad otto byte
                ' in una in doppia precisione.
```

Per economia e coerenza, QuickBASIC usa formati numerici standard (IEEE) per rappresentare interi, interi lunghi, valori a virgola mobile in doppia precisione ed in precisione semplice in un file ad accesso casuale. Se si usa l'istruzione FIELD per definire la struttura di un file ad accesso casuale, il programma deve esplicitamente convertire questi formati numerici in numeri dopo aver letto i dati dal file. Le funzioni CVI, CVL, CVS e CVD prendono variabili FIELD del formato del numero come argomenti e ritornano i valori numerici equivalenti. Ognuno di questi formati del numero ha una lunghezza fissa:

- Il formato intero richiede due byte.

- Il formato intero lungo richiede quattro byte.
- Il formato a precisione semplice richiede quattro byte.
- Il formato a precisione doppia richiede otto byte.

Un esempio

Il seguente programma illustra l'utilizzo di queste funzioni di conversione. Il programma legge i record da un file ad accesso casuale chiamato PROVA.DAT. (Un altro esempio di programma che crea questo file ad accesso casuale viene presentato nelle voci MKI$, MKL$, MKS$ e MKD$.) Poiché la struttura del record del file contiene un intero ed un campo a precisione semplice, il programma richiede l'uso di CVI e CVS per convertire questi campi in numeri:

```
' Lettura di un file ad accesso casuale: stile BASICA.

OPEN "PROVA.DAT" FOR RANDOM AS #1 LEN = 9
FIELD #1, 3 AS mese$, 2 AS unVend$, 4 AS entr$

DO
   CLS
   PRINT "Le vendite di quale mese"
   INPUT "vuoi vedere? <1> a <12>: ", me%
IF me% >= 1 AND me% <= 12 THEN
   GET #1, me%

   PRINT : PRINT
   PRINT "Mese:  "; mese$;
   IF me% = 5 THEN PRINT ELSE PRINT "."
   PRINT USING "Unità vendute:  ###"; CVI(unVend$)
   PRINT USING "Incasso totale: $$##,###.##"; CVI(entr$)
END IF

   PRINT : PRINT
   PRINT "Un altro mese? ";
   LOCATE , , 1
   DO
      contin$ = UCASE$(INKEY$)
   LOOP WHILE contin$ = ""
   PRINT contin$

LOOP UNTIL contin$ = "N"

CLOSE #1
```

Il file PROVA.DAT contiene record delle vendite, un record per ogni mese dell'anno. Una volta che il file è aperto, l'utente ha l'opportunità di scegliere il record di un mese da visualizzare; il programma mostra su schermo il contenuto di questo record. Ecco un'elaborazione d'esempio del programma:

```
Le vendite di quale mese
vuoi vedere? <1> a <12>: 11

Mese:            Nov.
Unità vendute:   758
Incasso totale:  L. 12.290.000

Un altro mese? N
```

L'istruzione FIELD del programma esprime la struttura semplice del record del file:

```
FIELD #1, 3 AS mese$, 2 AS unVend$, 4 AS entr$
```

Il campo *mese$* è un valore stringa, ma i campi *unVend$* e *entr$* rappresentano valori numerici, rispettivamente un intero ed un valore a virgola mobile in precisione semplice. Dopo l'istruzione GET # del programma legge un record selezionato dal file, i due campi di formato numerico devono essere convertiti in numeri:

```
PRINT USING "Unità vendute:  ###"; CVI(unVend$)
PRINT USING "Incasso totale: $$##,###.##"; CVS(entr$)
```

Ecco la funzione CVI che fornisce l'equivalente intero del campo *unVend$* e la funzione CVS fornisce l'equivalente virgola mobile in precisione semplice del campo *entr$*.

Compatibilità

Turbo Basic ha tutte e quattro queste funzioni di conversione. BASICA, che non definisce interi lunghi, sta perdendo la funzione CVL. In entrambe queste versioni del linguaggio, l'istruzione FIELD è la sola tecnica disponibile per definire la struttura di un file ad accesso casuale.

CVSMBF, CVDMBF

Le funzioni CVSMBF e CVDMBF forniscono compatibilità con un formato numerico, chiamato "formato Binario Microsoft", che era usato per rappresentare numeri in virgola mobile nelle prime versioni del BASIC Microsoft. Ogni funzione prende come proprio argomento una stringa di formato binario e ritorna un valore numerico nel formato standard IEEE.

```
CVSMBF(strSing)      ' Converte una stringa binaria a quattro byte
                     ' in un valore a virgola mobile in precisione
                     ' semplice.

CVDMBF(strDop)       ' Converte una stringa binaria a otto byte
                     ' in un valore a virgola mobile in precisione
                     ' doppia.
```

Si possono usare queste due funzioni per leggere dati in virgola mobile, da file ad accesso casuale creati con le prime versioni del BASIC Microsoft (QuickBASIC 2.0 e precedenti, o BASICA). La funzione CVSMBF prende una stringa del campo con formato binario a quattro byte come proprio argomento e ritorna un numero in precisione semplice. CVDMBF prende una stringa del campo con formato binario a otto byte come proprio argomento e ritorna un numero in precisione doppia.

Esempio

Il seguente frammento di programma illustra la tecnica per leggere numeri in formato binario da un file ad accesso casuale. Per prima cosa un programma deve definire un campo stringa che è della lunghezza appropriata per memorizzare un numero con formato binario letto dal file. Ad esempio, la seguente struttura definita dall'utente comprende un campo stringa chiamato *entr* che contiene i quattro byte necessari per un numero binario in precisione semplice:

```
TYPE tipMese
   mese AS STRING * 3
   unVend AS INTEGER
   entr AS STRING * 4
END TYPE

DIM recMese AS tipMese
```

Poi, il programma apre il file e legge un record da esso. Ad esempio, l'istruzione GET # legge qui di seguito un record dalla posizione *me%*:

```
OPEN "PROVA.dat" FOR RANDOM AS #1 LEN = LEN(recMese)
' ...
GET #1, me%, recMese
```

Infine, il programma usa CVSMBF o CVDMBF per convertire il campo stringa con formato binario in un numero con formato IEEE:

```
PRINT USING "Incasso totale: $$##,###.##"; CVSMBF(recMese.entr)
```

Si noti che i valori interi presi da un vecchio file ad accesso casuale non richiedono un trattamento speciale.

Compatibilità

Il Turbo Basic usa il formato IEEE per rappresentare numeri in virgola mobile; esso contiene funzioni chiamate CVMS e CVMD per convertire il formato Binario Microsoft nel formato IEEE.

FIELD

L'istruzione FIELD descrive la struttura del record di un file ad accesso casuale ed assegna una lunghezza fissa per ogni campo.

```
FIELD #num, lungh AS camp, lungh AS camp, ...
```

L'argomento *#num* è il numero di file assegnatogli da una precedente istruzione OPEN. In ogni descrizione del campo, *lungh* è un intero che indica il numero di byte nel campo, e *camp* è una variabile stringa che rappresenta lo spazio della memoria allocato nel campo. La lunghezza totale di tutti i campi di solito è la stessa di quella definita dal record del file, come specificato nella clausola LEN dell'istruzione OPEN che aveva aperto all'inizio il file. Quando si apre un file di dati, QuickBASIC alloca un'area speciale del *buffer* nella memoria per memorizzare i dati che sono sul tragitto verso o dal file. La clausola LEN nell'istruzione OPEN determina la

lunghezza del buffer e l'istruzione FIELD alloca porzioni del buffer a lunghezza fissa in particolari campi. Non si possono fare assegnamenti diretti a questa area del buffer nello stesso modo in cui si assegna un valore ad una variabile. Invece, si usa l'istruzione LSET o RSET per copiare un valore in una variabile del campo stringa definito nell'istruzione FIELD. Inoltre, si devono convertire i numeri in speciali stringhe formattate prima di poter copiare i valori delle variabili del campo. (Eseguono questa conversione le funzioni MKI$, MKL$, MKS$ e MKD$.) L'istruzione FIELD deve allocare la corretta quantità di spazio per i campi di tipo numerico:

- due byte per i valori interi,
- quattro byte per i valori interi lunghi,
- quattro byte per i valori in precisione semplice,
- otto byte per i valori in precisione doppia.

Per convenienza si possono scrivere più istruzioni FIELD per un singolo file aperto, definendo modi diversi per leggere o scrivere nella stessa area del buffer.

Un esempio di FIELD

Il programma presentato nelle voci CVI, CVS, CVL e CVD lavora con un file ad accesso casuale chiamato PROVA.DAT. Il file contiene un campo stringa, un campo intero ed uno in virgola mobile a precisione semplice. Queste istruzioni aprono il file e definiscono la struttura del record:

```
OPEN "PROVA.DAT" FOR RANDOM AS #1 LEN = 9
FIELD #1, 3 AS mese$, 2 AS unVend$, 4 AS entr$
```

Dopo che un'istruzione GET # ha letto dal file un record selezionato, il programma accede ai valori del campo direttamente dalle variabili del campo:

```
GET #1, me%
```

```
PRINT : PRINT
PRINT "Mese:   "; mese$;
IF me% = 5 THEN PRINT ELSE PRINT "."
PRINT USING "Unità vendute:  ###"; CVI(unVend$)
PRINT USING "Incasso totale: $$##,###.##"; CVI(entr$)
```

Si noti che i campi *unVend$* e *entr$* devono essere convertiti in valori numerici, rispettivamente da CVI e CVS. Nello stile QuickBASIC della programmazione di file di dati ad accesso casuale, si userebbe invece una variabile record come mezzo per leggere i dati dal file. Ecco come potrebbe presentarsi lo stesso programma:

```
' Lettura di un file ad accesso casuale: stile QuickBASIC.

TYPE tipMese
   mese AS STRING * 3
   unVend AS INTEGER
   entr AS SINGLE
END TYPE

DIM recMese AS tipMese

OPEN "PROVA.DAT" FOR RANDOM AS #1 LEN = LEN(recMese)

DO
   CLS
   PRINT "Le vendite di quale mese"
   INPUT "vuoi vedere? <1> a <12>: ", me%

   IF me% >= 1 AND me% <= 12 THEN
      GET #1, me%, recMese

      PRINT : PRINT
      PRINT "Mese:   "; recMese.mese;
      IF me% = 5 THEN PRINT ELSE PRINT "."
      PRINT USING "Unità vendute:  ###"; recMese.unVend
      PRINT USING "Incasso totale: $$##,###.##"; recMese.entr
   END IF
   PRINT : PRINT
   PRINT "Un altro mese? ";
   LOCATE , , 1
   DO
      contin$ = UCASE$(INKEY$)
   LOOP WHILE contin$ = ""
   PRINT contin$


LOOP UNTIL contin$ = "N"

CLOSE #1
```

Compatibilità

Sia Turbo Basic sia BASICA forniscono l'istruzione FIELD per definire la struttura del record di un file ad accesso casuale. Nessuna versione del linguaggio supporta l'uso di variabili record.

LSET E RSET

Le istruzioni LSET e RSET copiano valori nelle variabili FIELD definite per un file ad accesso casuale.

```
LSET nomCamp = val   ' Dà al valore la giustificazione a
                     ' sinistra nella lunghezza del campo.

RSET nomCamp = val   ' Dà al valore la giustificazione a
                     ' destra nella lunghezza del campo.
```

Quando si crea un nuovo file ad accesso casuale, QuickBASIC alloca un'area speciale del *buffer* nella memoria per memorizzare i dati che stanno per essere scritti nel file. La clausola LEN nell'istruzione OPEN determina la lunghezza del buffer e l'istruzione FIELD alloca porzioni del buffer a lunghezza fissa in particolari campi. Non si possono fare assegnamenti diretti a questa area del buffer nello stesso modo in cui si assegna un valore ad una variabile. Invece, si usa l'istruzione LSET o RSET per copiare un valore in una variabile del campo stringa definito nell'istruzione FIELD. Se il valore è inferiore alla lunghezza del campo, LSET dà la giustificazione a sinistra e RSET la giustificazione a destra al valore del campo; la lunghezza rimanente viene colmata con degli spazi. I valori numerici devono essere convertiti in formati standard di memoria a lunghezza fissa, prima di essere copiati nel buffer del file. Eseguono questa conversione le funzioni MKI$, MKL$, MKS$ e MKD$.

Altri usi

Sia LSET sia RSET hanno utilizzi secondari che non hanno niente a che fare con i file ad accesso casuale. Ad esempio, si può usare RSET per giustificare a destra un valore stringa all'interno della lunghezza attuale di una variabile

stringa. Inoltre, la funzione LSET può essere usata per copiare il contenuto di una variabile record in un'altra che appartiene ad un diverso tipo definito dall'utente; comunque, questa operazione richiede una certa attenzione e non può essere eseguita a caso.

Un esempio di LSET

Il programma presentato nelle voci MKI$, MKL$, MKS$ e MKD$ crea un file ad accesso casuale chiamato PROVA.DAT. Il file contiene tre campi: una stringa, un intero ed un valore in virgola mobile a precisione semplice. Per ogni record, il programma legge i valori da una linea DATA e poi usa tre istruzioni LSET per copiare i dati nelle variabili del campo. Infine, un'istruzione PUT# scrive il record nel file:

```
READ nomMese$, unVend%, entr
LSET mese$ = nomMese$
LSET unVend$ = MKI$(unVend%)
LSET entr$ = MKS$(entr)
PUT #1, me%
```

Nello stile QuickBASIC della programmazione del file di dati ad accesso casuale, si userebbe invece una variabile record come mezzo per scrivere i dati nel file. Ad esempio, ecco come potrebbe presentarsi lo stesso programma nello stile QuickBASIC:

```
' Creazione di un file ad accesso casuale: stile QuickBASIC.

TYPE tipMese
   mese AS STRING * 3
   unVend AS INTEGER
   entr AS SINGLE
END TYPE

DIM recMese AS tipMese

OPEN "PROVA.DAT" FOR RANDOM AS #1 LEN = 9

FOR me% = 1 TO 12
   READ recMese.mese
   READ recMese.unVend
   READ recMese.entr
   PUT #1, me%, recMese
```

```
NEXT me%

CLOSE #1

DATA Gen, 344, 5.487.000
DATA Feb, 255, 4.068.000
DATA Mar, 644, 10.272.000
DATA Apr, 497, 7.928.000
DATA Mag, 688, 10.975.000
DATA Giu, 219, 3.493.000
DATA Lug, 448, 7.145.000
DATA Ago, 556, 8.870.000
DATA Set, 498, 7.944.000
DATA Ott, 710, 11.325.000
DATA Nov, 758, 12.100.000
DATA Dic, 801, 12.776.000
```

Compatibilità

Sia Turbo Basic sia BASICA forniscono le istruzioni LSET e RSET per copiare i valori delle variabili del campo di un file ad accesso casuale. Nessuna versione del linguaggio supporta l'uso di variabili record.

MKI$, MKL$, MKS$, MKD$

Dati degli argomenti numerici appropriati, le funzioni MKI$, MKL$, MKS$ e MKD$ forniscono i formati standard numerici a lunghezza fissa che QuickBASIC usa nei file ad accesso casuale.

```
MKI$(int)   ' Converte un intero in una stringa a due byte.

MKL$(lung)  ' Converte un intero lungo in una stringa a quattro byte.

MKS$(semp)  ' Converte un valore in virgola mobile a precisione
            ' semplice in una stringa a quattro byte.

MKD$(dop)   ' Converte un valore in virgola mobile a doppia precisione
            ' in una stringa a otto byte.
```

Per economia e comodità, QuickBASIC usa formati numerici standard (IEEE) per rappresentare interi, interi lunghi, valori in virgola mobile a precisione semplice e doppia in un file di dati ad accesso casuale. Se si usa

l'istruzione FIELD per definire la struttura di un file ad accesso casuale, si devono esplicitamente convertire i valori numerici in questi formati prima di scrivere i dati nel file. Le funzioni MKI$, MKL$, MKS$ e MKD$ prendono argomenti numerici e ritornano stringhe standard di formato numerico che si possono copiare nelle variabili FIELD. Ogni formato numerico ha una lunghezza fissa:

- Il formato intero richiede due byte.
- Il formato intero lungo richiede quattro byte.
- Il formato a precisione semplice richiede quattro byte.
- Il formato a precisione doppia richiede otto byte.

Un esempio

Questo programma illustra le funzioni di conversione. Crea un file ad accesso casuale chiamato PROVA.DAT. (Un esempio del programma presentato nella voce CVI, CVL, CVS, CVD è progettato per leggere il file PROVA.DAT.) Poiché la struttura record del file contiene un campo intero ed uno a precisione semplice, il programma usa MKI$ e MKS$ per convertire valori numerici in appropriate stringhe di formato numerico:

```
' Creazione di un file ad accesso casuale: stile BASICA.

OPEN "PROVA.DAT" FOR RANDOM AS #1 LEN = 9
FIELD #1, 3 AS mese$, 2 AS unVend$, 4 AS entr$
FOR me% = 1 TO 12
   READ nomMese$, unVend%, entr
   LSET mese$ = nomMese$
   LSET unVend$ = MKI$(unVend%)
   LSET entr$ = MKS$(entr)
   PUT #1, me%
NEXT me%

CLOSE #1

DATA Gen, 344, 5.487.000
DATA Feb, 255, 4.068.000
DATA Mar, 644, 10.272.000
DATA Apr, 497, 7.928.000
DATA Mag, 688, 10.975.000
```

```
DATA Giu, 219, 3.493.000
DATA Lug, 448, 7.145.000
DATA Ago, 556, 8.870.000
DATA Set, 498, 7.944.000
DATA Ott, 710, 11.325.000
DATA Nov, 758, 12.100.000
DATA Dic, 801, 12.776.000
```

L'istruzione FIELD del programma esprime la struttura record del file:

```
FIELD #1, 3 AS mese$, 2 AS unVend$, 4 AS entr$
```

Il campo *mese$* è un valore stringa, *unVend$* rappresenta un intero e *entr$* rappresenta un valore in virgola mobile a precisione semplice. Il programma legge i dati da ogni record del file tramite una sequenza di istruzioni DATA. Prima che i due campi numerici possano essere scritti nel file, il programma usa MKI$ e MKS$ per convertirli in stringhe con formato numerico, che possono poi essere copiate nelle variabili del campo:

```
LSET unVend$ = MKI$(unVend%)
LSET entr$ = MKS(entr)
```

Una volta che tutti e tre i valori sono stati copiati nelle variabili del campo, il programma scrive il record corrente nel file:

```
PUT #1, me%
```

Compatibilità

Turbo Basic ha tutte e quattro queste funzioni di conversione. BASICA, che non definisce gli interi lunghi non ha MKL$. In entrambe queste versioni del linguaggio, l'istruzione FIELD è la sola tecnica disponibile per definire la struttura di un file ad accesso casuale.

MKSMBF$, MKDMBF$

Le funzioni MKSMBF$ e MKDMBF$ danno la compatibilità con un formato numerico, chiamato "formato Binario Microsoft", che era usato per

rappresentare i valori in virgola mobile nelle prime versioni del BASIC Microsoft. Ogni funzione prende un numero con formato IEEE come proprio argomento e ritorna una rappresentazione a stringa con formato binario del numero.

```
MKSMBF$(semp)       ' Converte un valore IEEE in precisione semplice
                    ' in una stringa con formato binario a quattro byte.

MKDMBF$(dop)        ' Converte un valore IEEE in precisione doppia in una
                    ' stringa con formato binario a otto byte.
```

Si possono usare queste due funzioni per scrivere i dati in virgola mobile nei file ad accesso casuale che devono poter essere leggibili nelle prime versioni del BASIC Microsoft (QuickBASIC 2.0 e precedenti o BASICA). La funzione MKSMBF$ prende un numero con formato IEEE in precisione semplice come proprio argomento, e ritorna una stringa con formato binario a quattro byte. MKDMBF$ prende un numero con formato IEEE a doppia precisione come proprio argomento, e ritorna una stringa con formato binario a otto byte.

Un esempio

Questo stralcio di programma illustra la tecnica per scrivere i numeri in formato binario in un file ad accesso casuale che devono poter essere leggibili in vecchie versioni del BASIC Microsoft. Per prima cosa un programma definisce un campo stringa che è della lunghezza giusta per memorizzare il numero in formato binario che sta per essere scritto nel file. Ad esempio, questa struttura definita dall'utente comprende un camèo stringa chiamato *entr* che contiene i quattro byte necessari per un numero binario a precisione semplice:

```
TYPE tipMese
   mese AS STRING * 3
   unVend AS INTEGER
   entr AS STRING * 4
END TYPE

DIM recMese AS tipMese
```

Poi, il programma apre il file e forma un record di dati da inviare al file. Ad esempio, il seguente frammento legge tre valori del campo dalle linee DATA:

```
OPEN "PROVA.DAT" FOR RANDOM AS #1 LEN = 9

   READ recMese.mese
   READ recMese.unVend
   READ tempEntr
   recMese.entr = MKSMBF$(tempEntr)
```

Si noti che qui l'istruzione finale usa MKSMBF$ per convertire il numero in precisione semplice *tempEntr* in un campo stringa con formato binario *recMese.entr*. (Non è richiesto nessun trattamento speciale per il campo intero.) Infine, il programma scrive il record nel file:

```
PUT #1, me%, recMese
```

Compatibilità

Turbo Basic usa il formato IEEE per rappresentare i numeri in virgola mobile; contiene funzioni chiamate MKMS$ e MKMD$ per convertire numeri con formato IEEE in campi stringa con formato binario.

Capitolo 13

Altri dispositivi d'input ed output

Questo capitolo prende in considerazione le tecniche d'input ed output per una vasta selezione di dispositivi. Gli strumenti qui trattati comprendono:

- Funzioni ed istruzioni che lavorano con dispositivi d'input ed output con compiti particolari come il joystick, la penna ottica e il modem connesso ad una porta seriale di comunicazione.
- Funzioni ed istruzioni che sono progettate per l'accesso diretto alle porte di I/O in contesti di programmazione a basso livello.

Le voci di questo capitolo forniscono solo brevi riassunti di questi comandi e funzioni. Per ulteriori informazioni, consultare i manuali tecnici sugli specifici sistemi hardware e strumenti.

INP E OUT

La funzione INP legge un byte di dati da una specifica porta hardware I/O, mentre l'istruzione OUT invia un byte di dati ad una porta.

```
INP(porta)        ' porta va da 0 a 65.535.
OUT porta, byte   ' byte va da 0 a 255.
```

La funzione INP e l'istruzione OUT sono strumenti I/O di basso livello che danno l'accesso diretto alle porte.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione INP e l'istruzione OUT.

IOCTL$ E IOCTL

La funzione IOCTL$ ritorna una stringa di informazioni di controllo ricevute da un driver attivo. L'istruzione IOCTL passa le informazioni di controllo ad un driver attivo.

```
IOCTL$(#num)      ' num è il numero di un file aperto.

IOCTL #num, str   ' str sono le informazioni di controllo
                  ' inviate al driver.
```

La funzione IOCTL$ e l'istruzione IOCTL danno un accesso diretto del programma di QuickBASIC ad un driver installato, ammettendo che il driver supporti le chiamate IOCTL. L'argomento *#num* si riferisce al numero di un file aperto per il dispositivo in questione. L'utilizzo di questi due strumenti richiede informazioni dettagliate sul funzionamento del driver interessato.

Compatibilità

Turbo Basic e le versioni recenti del BASIC Microsoft supportano la funzione IOCTL$ e l'istruzione IOCTL.

OPEN COM

L'istruzione OPEN COM tenta di stabilire un collegamento di comunicazione seriale con specifiche opzioni.

```
OPEN "COMn: opzioni" FOR modo AS #num LEN=lungh     ' Le clausole FOR
                                                    ' e LEN sono
                                                    ' facoltative.
```

OPEN COM identifica una porta I/O che verrà usata per le comunicazioni seriali e stabilisce le caratteristiche di trasmissione. La porta di comunica-

zione viene identificata come COM1: o COM2:. La clausola opzionale FOR dà il modo per il file di comunicazione aperto: INPUT, OUTPUT o RANDOM. Il default è RANDOM, il modo che consente sia l'input sia l'output. La clausola AS specifica il numero del file (da 1 a 255) che verrà usato per identificare il file nelle successive istruzioni di input ed output. Infine, la clausola LEN stabilisce la lunghezza del buffer associato al file se il file di comunicazione è aperto nel modo RANDOM. La lunghezza di default è 128 byte. La stringa di opzioni listata dopo il nome della porta deve contenere nel seguente ordine:

baud rate, parità, bit di dati, bit di stop, altre opzioni

Ogni opzione è separata dalla successiva da una virgola, e, se ne viene omessa una, la virgola deve fare da segnaposto:

- Il baud rate di default è 300. Altri valori ammessi sono 75, 110, 150, 1200, 1800, 2400 e 9600.
- L'opzione di parità di default è E per pari. Altre sono N per nessuna, O per dispari, S per spazio e M per punto.
- Il numero di default di bit dei dati per byte è 7. Il numero di bit dei dati può essere un intero che va da 5 a 8.
- Il numero di default dei bit di stop è 1. I bit di stop possono essere anche 2.
- Le altre opzioni comprendono delle combinazioni di: ASC (modo ASCII), BIN (modo binario), CD*m* (rilevazione del timeout della portante, dove *m* è in millisecondi), CS*m* (rilevazione del timeout sulla linea clear to send), DS*m* (rilevazione del timeout sulla linea data set ready), LF (carattere di fine riga aggiunto a quello di ritorno carrello), OP*m* (tempo d'attesa delle comunicazioni), RB*n* e TB*n* (dimensioni dei buffer riceventi e trasmettitori, dove *n* è in byte) e RS (non riconosce la transizione sulla linea request to send).

Un esempio di OPEN COM

La seguente istruzione apre un collegamento seriale come file #3 sulla porta 2, usando un baud rate di 600 e i default per tutte le altre opzioni:

```
OPEN "COM2:600" AS #3
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione OPEN COM.

PEN

La funzione PEN ritorna informazioni sull'attività della penna ottica.

```
PEN(num)     ' num è un'opzione che va da 0 a 9.
```

L'argomento numerico seleziona una delle dieci informazioni che la funzione PEN dà sullo stato della penna ottica:

0 Ritorna il valore logico vero se la penna è stata spenta dall'ultima chiamata e falso in caso contrario.

1 Dà le coordinate *x* sullo schermo grafico della precedente attivazione della penna.

2 Dà le coordinate *y* sullo schermo grafico della precedente attivazione della penna.

3 Ritorna il valore logico vero se la penna al momento è spenta e falso in caso contrario.

4 Dà le coordinate *x* sullo schermo grafico della precedente posizione della penna.

5 Dà le coordinate *y* sullo schermo grafico della precedente posizione della penna.

6 Dà la riga di testo della precedente attivazione della penna.

7 Dà la colonna di testo della precedente attivazione della penna.

8 Dà la riga di testo della precedente posizione della penna.

9 Dà la colonna di testo della precedente posizione della penna.

Le istruzioni ON PEN e PEN ON/OFF/STOP definiscono ed attivano una gestione degli eventi per l'attività della penna ottica. Non si possono usare contemporaneamente una penna ottica ed un mouse.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione PEN.

STICK

La funzione STICK identifica le posizioni correnti di due joystick.

```
STICK(num)     ' num è un'opzione che va da 0 a 3.
```

Le coordinate *x* ed *y* del joystick A sono date da STICK(0) e STICK(1). Allo stesso modo, le coordinate del joystick B sono STICK(2), STICK(3). Una chiamata a STICK(0) legge tutte e quattro le coordinate e deve essere eseguita per prima. Le altre opzioni *num* ritornano le coordinate del joystick come erano al momento della prima chiamata di STICK(0).

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione STICK.

STRIG

La funzione STRIG ritorna un valore logico che indica se un particolare pulsante del joystick è stato premuto o meno.

```
STRIG(num)     ' num è un'opzione che va da 0 a 7.
```

L'argomento numerico indica un pulsante su uno dei due joystick. Le opzioni 0 e 1 rilevano lo stato del pulsante inferiore sul joystick A, mentre le opzioni 4 e 5 quello del pulsante superiore. Allo stesso modo le opzioni

2 e 3 sono per il pulsante inferiore del joystick B e le opzioni 6 e 7 per il pulsante B superiore. Dati gli argomenti di 1, 3, 5 o 7, STRIG ritorna il valore logico vero se il pulsante specifico al momento è abbassato e falso se non lo è. Per gli argomenti di 0, 2, 4 o 6, STRIG è il valore logico vero se il pulsante è stato premuto dall'ultima chiamata o falso in caso contrario.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione STRIG.

WAIT

L'istruzione WAIT legge continuamente un valore di un byte da una specifica porta di I/O fino a che il valore corrisponde ad una particolare sequenza di bit.

```
WAIT port, byte1, byte2   ' byte2 è opzionale.
```

L'argomento *port* è un intero che va da 0 a 65.535 e *byte1* e *byte2* sono interi che vanno da 0 a 255. WAIT legge un valore *n* dalla porta specifica ed esegue la seguente operazione bit:

```
(n XOR byte2) AND byte1
```

Se il risultato è un valore logico vero (diverso da 0) WAIT lascia il controllo all'istruzione successiva del programma. Se, però, il risultato è un valore logico falso (0), WAIT legge un altro valore byte dalla porta ed esegue ancora il test bit. In effetti, l'esecuzione del programma viene sospesa finché l'operazione bit dà il valore logico vero. Se si omette *byte2*, WAIT usa un valore di default pari a 0 per l'operando XOR. (Il risultato di *n* XOR 0 è sempre *n*.)

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione WAIT.

Capitolo 14

Suono

QuickBASIC fornisce degli strumenti che producono dei segnali sonori e note musicali dall'altoparlante nel computer. Queste istruzioni si differenziano per complessità, versatilità e potenza:

- L'istruzione BEEP genera semplicemente una sola nota breve.
- L'istruzione SOUND crea un suono caratterizzato dalla propria frequenza e durata.
- L'istruzione PLAY esegue una sequenza di note musicali.

Queste istruzioni (insieme alla funzione chiamata PLAY) sono l'argomento di questo capitolo.

BEEP

L'istruzione BEEP fa sì che l'altoparlante del computer produca una singola nota breve.

```
BEEP
```

BEEP non ha argomenti. L'effetto è uguale fornendo il carattere ASCII del cicalino in un'istruzione PRINT:

```
PRINT CHR$(7)
```

Un esempio di BEEP

BEEP è un modo semplice per attirare l'attenzione dell'utente nel momento in cui un programma visualizza sullo schermo alcune importanti informazioni. Questo frammento di codice visualizza un messaggio d'errore e produce un segnale sonoro contemporaneamente:

```
BEEP
PRINT "Non si trova il file sul corrente disco di default."
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione BEEP. In Turbo Basic, BEEP accetta un argomento numerico opzionale che specifica il numero di beep che deve emettere l'altoparlante. Per implementare questa modalità in QuickBASIC, si può usare il seguente sottoprogramma:

```
SUB PiùBeep (num%)
   FOR i% = 1 TO num%
      BEEP
   NEXT i%
END SUB
```

Ad esempio, questa istruzione di chiamata dà cinque beep in una riga:

```
PiùBeep 5
```

LA FUNZIONE PLAY

La funzione PLAY evidenzia un intero che indica il nunero di note attualmente disponibili per eseguire musiche di sottofondo (MB) dell'istruzione PLAY.

```
PLAY(num)       ' num non ha alcun significato.
```

L'argomento numerico fittizio della funzione PLAY può essere un valore qualsiasi. Si può usare PLAY() in una routine di gestione degli eventi ON PLAY per determinare il numero di note attualmente nel buffer per la musica di sottofondo.

Un esempio di PLAY

Il seguente esercizio è una semplice dimostrazione della funzione PLAY. Il programma predispone una sequenza di note da suonare in sottofondo. Poi, mentre le note vengono eseguite, il programma visualizza sullo schermo il valore di PLAY:

```
strSuo$ = "MB"
FOR i% = 21 TO 52
   strSuo$ = strSuo$ + "N" + STR$(i%)
NEXT i%

PLAY strSuo$

CLS
DO
   LOCATE 10, 10
   PRINT USING "Note rimaste: ###"; PLAY(0)
LOOP UNTIL PLAY(0) = 0
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione PLAY.

L'ISTRUZIONE PLAY

L'istruzione PLAY esegue una sequenza di note musicali tramite l'altoparlante del computer.

```
PLAY stringa
```

L'argomento *stringa* contiene la combinazione di comandi PLAY di una lettera che specificano le note musicali e le loro caratteristiche:

- Le lettere A, B, C, D, E, F e G rappresentano le note della scala musicale. Si può aggiungere il carattere #(diesis) per una nota più acuta di un semitono o il carattere -(bemolle) per una nota grave. Sono disponibili sette ottave. Il comando O*n* stabilisce l'ottava corrente ed *n* è un intero che va da 0 a 6. L'ottava di default è O4. Si possono usare anche il comando

> per alzarsi di un'ottava ed il comando < per abbassarsi di un'ottava. (Un altro modo per eseguire delle note è usare il comando N*n* dove *n* è un intero che va da 1 a 84 e rappresenta una delle note fra le sette ottave disponibili. Ad esempio, N49 è uguale a C nell'ottava di default: O4.)

- Il comando L*n* stabilisce la lunghezza delle note che seguono. Il valore di *n* è un intero che va da 1 a 64. Ad esempio, L1 dà le note piene, L2 i semitoni, L4 i quarti di tono, per default. Si può stabilire anche la lunghezza di una singola nota includendo un intero che va da 1 a 64 direttamente dopo la nota. Inoltre un punto dopo una nota ne aumenta la lunghezza del 50 per cento. Il comando P*n* esegue una pausa e *n* è un intero che va da 1 a 64. P1 è una pausa piena, P2 una mezza pausa e così via. Il comando T*n* stabilisce il tempo, cioè il numero di quarti di tono al minuto, e *n* è un intero che va da 32 a 255. Il default è T120. I comandi MN (modo normale), ML (modo legato) e MS (modo staccato) controllano le transizioni tra note; normale e staccato risultano nei mutamenti di registro tra note, mentre legato produce una transizione uguale senza una pausa.
- Il comando MF esegue la musica in primo piano, modo default. MB esegue la musica in sottofondo così che altre istruzioni possono essere eseguite mentre si suona la musica. (Sia l'istruzione ON PLAY sia la funzione PLAY consentono di scrivere dei programmi che controllano esplicitamente lo stato del buffer per la musica di sottofondo.)
- Il comando X esegue una sottostringa di note musicali. Per gestire con successo una sottostringa di questo tipo in un programma compilato si usa la funzione VARPTR$.

I comandi nella stringa PLAY possono essere scritti in lettere maiuscole o minuscole e se ne possono includere tanti quanti si vuole.

Un esempio di PLAY

Il programma *DateStoriche,* listato nel Capitolo 32, è un gioco a quiz che mette alla prova la conoscenza del giocatore per quanto riguarda le date storiche. Il programma usa il comando PLAY per emettere una o due brevi sequenze di note ogni volta che il giocatore risponde ad una domanda. Una è per le risposte corrette e l'altra per quelle sbagliate. Ecco il frammento che seleziona una sequenza PLAY e la esegue:

```
IF esat% THEN
   PLAY "o4 l8 e l16 f# e > l3 c"
ELSE
   PLAY "o2 l8 e l16 f# e < l3 c"
END IF
```

Le due sequenze sono simili, ma sono suonate in ottave diverse. Il primo tema sale di un'ottava alla fine, mentre il secondo scende di un'ottava.

Compatibilità

Sia Turbo Basic sia BASICA hanno il comando PLAY.

L'ISTRUZIONE SOUND

Con l'istruzione SOUND l'altoparlante del computer produce un suono caratterizzato da una specifica frequenza e durata.

```
SOUND hertz, tick
```

I due argomenti numerici dell'istruzione SOUND possono apparire come valori literal, variabili o espressioni. Il primo argomento specifica la frequenza desiderata del suono emesso in hertz (cicli al secondo). L'intervallo deve essere compreso tra 37 e 32.767, sebbene l'udito umano finisca tra i 14.000 e i 15.000 cicli al secondo. Il secondo argomento è la durata del suono in *tick*, che corrisponde circa a un ventesimo di secondo. (Più precisamente ci sono 18,2 tick al secondo.) Il numero massimo possibile di tick è 65.535, che equivale ad un'ora.

Un esempio di SOUND

Il programma *DateStoriche,* listato nel Capitolo 32, è un gioco a quiz che mette alla prova la conoscenza del giocatore per quanto riguarda le date storiche. Per misurare il tempo di risposta dell'utente alle domande, il programma crea sullo schermo un timer ad orologio con una lancetta che conta i secondi. Un giro dell'orologio rappresenta il tempo impiegato per

una data domanda. Per rendere questo orologio più realistico, il programma *DateStoriche* ha anche degli effetti sonori. Ogni volta che la lancetta si sposta in avanti di un secondo, il programma emette un breve tick. Se nel corso del giro completo dell'orologio il giocatore non ha risposto alla domanda, il programma emette un suono più acuto che indica al giocatore che il suo tempo a disposizione è finito. Questi effetti sonori vengono prodotti in una routine di gestione degli eventi chiamata *ClicSeg*, che viene chiamata dal programma ogni secondo per sistemare l'orologio. Ecco il frammento che utilizza l'istruzione SOUND:

```
IF ang% > fineAng% THEN
   IF NOT muto% THEN SOUND 50, .5
   CIRCLE (0, 0) .25, 1
   PAINT (0, 0), 1, 1
ELSE
   IF NOT muto% THEN SOUND 1000, 10
   TIMER OFF
   tempFin% = vero
END IF
```

Il rumore del ticchettio ha una bassa frequenza di 50 tick e una durata di solo mezzo tick:

```
SOUND 50, .5
```

Al contrario, il suono d'allarme ha una frequenza più alta ed una durata di mezzo secondo:

```
SOUND 1000, 10
```

Si noti che il programma produce questi effetti sonori solo se la variabile logica *muto%* è falsa. Il programma dà all'utente l'opportunità di giocare in modo "muto", cioè senza che sia emesso nessun suono. Questa è una caratteristica importante del progetto in ogni programma che usa le istruzioni SOUND, PLAY o BEEP per creare effetti sonori. Un utente dovrebbe avere sempre la possibilità di disattivare questi suoni.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione SOUND.

Parte IV

Grafica

Le applicazioni grafiche regalano al programmatore alcune delle più soddisfacenti sfide algoritmiche. Creare complesse immagini grafiche sullo schermo richiede sia abilità di calcolo sia una particolare attenzione. Fortunatamente, QuickBASIC fornisce un insieme di potenti strumenti grafici che consentono di controllare l'hardware grafico dello schermo e semplificare le tecniche di programmazione grafica.

La Parte IV descrive le istruzioni e le funzioni QuickBASIC che controllano i modi di visualizzazione dello schermo, i colori e le figure. Naturalmente, le possibilità offerte da questi strumenti dipendono dalla configurazione dell'hardware che costituisce lo schermo. Gli schermi grafici standard disponibili per i personal computer IBM e compatibili comprendono: l'Adattatore Grafico a Colori (CGA), l'Adattatore Grafico Avanzato (EGA), il VGA ed il MCGA. Questi schermi offrono una gamma di crescenti possibilità ed opzioni nel campo della programmazione grafica.

Per mostrare alcune tecniche di programmazione grafica e per illustrare alcuni strumenti grafici di QuickBASIC, nel Capitolo 32 viene presentato il programma *DateStoriche*. È un semplice gioco a quiz e presenta sullo schermo diversi grafici ed immagini durante il gioco. Molte delle singole voci dei Capitoli 15, 16 e 17 utilizzano estratti di questo programma come esempi di specifiche tecniche di programmazione grafica.

Capitolo 15

Schermi e finestre

QuickBASIC supporta l'intera serie di opzioni hardware dello schermo disponibili per i PC IBM e compatibili. Diversi schermi supportano specifici modi testo e schermo grafico ed ogni modo ha le proprie caratteristiche. L'istruzione SCREEN è lo strumento primario per ottenere l'accesso a questi modi. Questo capitolo descrive dettagliatamente SCREEN e le numerose altre istruzioni e funzioni collegate all'utilizzo degli schermi grafici:

- Le istruzioni VIEW e WINDOW insieme consentono di creare un appropriato sistema di coordinate per un dato modo schermo grafico.
- L'istruzione WIDTH controlla il numero di colonne dei caratteri e di righe del testo visualizzato in un modo schermo.
- Le funzioni POINT, PMAP e SCREEN danno le informazioni sul contenuto corrente dello schermo.
- L'istruzione PCOPY copia il contenuto di una *pagina* dello schermo in un'altra.

Questo capitolo fornisce molti brevi programmi di dimostrazione per illustrare questi strumenti e frammenti del programma *DateStoriche*, che viene discusso dettagliatamente nel Capitolo 32.

PCOPY

L'istruzione PCOPY copia uno schermo grafico da una pagina all'altra nel modo SCREEN corrente.

```
PCOPY pag1, pag2     ' pag1 è la sorgente;
                     ' pag2 è la destinazione.
```

Gli argomenti *pag1* e *pag2* sono interi che vanno da 0 fino al numero di pagine disponibili nel modo schermo corrente. PCOPY copia il contenuto di *pag1* in *pag2*. L'istruzione SCREEN consente di commutare tra visualizzazioni di diverse pagine e specificare quale tra le tante pagine riceverà l'output dalle istruzioni grafiche. Per ulteriori dettagli consultare la voce SCREEN.

Un esempio di PCOPY

Questo programma è progettato per dimostrare l'istruzione PCOPY. Il programma inizia disegnando una semplice immagine sulla pagina 0 nel modo SCREEN 9 ed utilizza PCOPY per copiare il disegno incompleto nella pagina 1. Poi il programma completa il disegno su entrambe le pagine in modi leggermente diversi. Infine, un ciclo DO commuta ripetutamente tra le visualizzazioni delle due pagine, creando un semplice effetto di animazione sullo schermo:

```
' dimostrazione di PCOPY.
pi = 4 * ATN(1)
SCREEN 9
COLOR 1, 15
WINDOW (-100, -100)-(100, 100)

LINE (80, 80)-(-80, -80), 4, BF
CIRCLE (0, 0), 50, 15
PAINT (0, 0), 15
CIRCLE (-20, 30), 5, 1
PSET (-20, 30), 1
CIRCLE (20, 30), 5, 1
PSET (20, 30), 1

PCOPY 0, 1
```

```
CIRCLE (0, 10), 40, 1, (4 / 3) * pi, (5 / 3) * pi
SCREEN , , 1
CIRCLE (0, -80), 30, 1, (1 / 3) * pi, (2 / 3) * pi

DO
   SCREEN , , , 1
   SLEEP 1
   SCREEN , , , 0
   SLEEP 1
LOOP UNTIL INKEY$ = " "
SCREEN 0
```

Premere la barra spaziatrice per terminare l'operazione di questo programma e ritornare alla visualizzazione del testo. Questo semplice esempio dimostra un inportante utilizzo di più pagine. È possibile disegnare figure simili su due o più pagine diverse e poi commutare velocemente in avanti o indietro tra le visualizzazioni della pagina per creare l'illusione di movimento sullo schermo.

Compatibilità

Né Turbo Basic né le prime versioni di BASICA hanno l'istruzione PCOPY. Comunque, l'istruzione è supportata nelle recenti versioni di GW-BASIC.

PMAP

La funzione PMAP converte una coordinata dell'indirizzo sull'area grafica, nell'equivalente coordinata dell'indirizzo sullo schermo oppure una coordinata dello schermo in una coordinata del grafico.

```
PMAP(coord, opz)     ' opz è un intero che va da 0 a 3 e
                     ' rappresenta un'opzione di conversione.
```

Dato un grafico ed un sistema di coordinate stabilite dalle istruzioni VIEW e/o WINDOW, la funzione PMAP è uno strumento che aiuta a convertire tra queste coordinate del grafico e le equivalenti coordinate dello schermo. Il primo argomento PMAP è un intero che rappresenta una coordinata orizzontale o verticale di un indirizzo. Il secondo argomento è un intero che va da 0 a 3, selezionando una delle opzioni di conversione:

0 Ritorna lo schermo equivalente di una coordinata *x* del grafico.

1 Ritorna lo schermo equivalente di una coordinata *y* del grafico.

2 Ritorna l'area grafica equivalente di una coordinata *x* dello schermo.

3 Ritorna l'area grafica equivalente di una coordinata *y* dello schermo.

Se si stanno convertendo le coordinate all'interno dell'area grafica, si deve modificare il risultato di PMAP con il valore dello spiazzamento precisato nell'istruzione VIEW per trovare gli indirizzi equivalenti dello schermo.

Un esempio di PMAP

Il programma seguente è progettato per dimostrare la funzione PMAP. Il programma stabilisce due aree grafiche nel modo schermo SCREEN 9 e disegna un cerchio all'interno di ogni area grafica. Mentre una data area grafica è attiva, il programma usa PMAP per trovare gli equivalenti indirizzi dello schermo del centro del cerchio. Poi dopo che entrambi i cerchi sono stati disegnati, il programma ripristina l'area grafica di default ed il sistema di coordinate e disegna una linea dal centro di un cerchio al centro di un altro. Gli indirizzi dello schermo di questa linea sono le coordinate che sono state calcolate con la funzione PMAP:

```
' dimostrazione della funzione PMAP.

DEFINT X-Y

SCREEN 9
COLOR 4, 15

VIEW (10, 10)-(210, 210), 4, 4
WINDOW (-10, -10)-(10, 10)
x1 = 10 + PMAP(0, 0)
y1 = 10 + PMAP(0, 1)
CIRCLE (0, 0), 8, 15

VIEW (250, 20)-(450, 320), 5, 5
WINDOW (-10, -10)-(10, 10)
x2 = 250 + PMAP(0, 0)
y2 = 20 + PMAP(0, 1)
CIRCLE (0, 0), 9, 15
```

```
VIEW
WINDOW
LINE (x1, y1)-(x2, y2), 15
```

I centri dei due cerchi sono rappresentati dagli indirizzi dello schermo *(x1, y1)* e *(x2, y2)*. Si noti che il programma aggiunge uno spiazzamento al valore di PMAP per calcolare questi indirizzi. In ogni caso lo spiazzamento è preso dall'indirizzo dello schermo nell'angolo in alto a sinistra dell'area grafica corrente:

```
x1 = 10 + PMAP(0, 0)
y1 = 10 + PMAP(0, 1)

x2 = 250 + PMAP(0, 0)
x2 = 20 + PMAP(0, 1)
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione PMAP.

POINT

Una sintassi della funzione POINT ritorna l'attributo del colore dell'indirizzo di un dato pixel sul corrente schermo grafico. La seconda sintassi aiuta a convertire un indirizzo dell'area grafica in un indirizzo dello schermo o viceversa.

```
POINT(x, y)     ' Dà il colore del pixel a (x, y).

POINT(opz)      ' Ritorna una coordinata dell'ultimo punto
                ' a cui si è fatto riferimento sullo schermo.
```

Data una coppia di coordinate come argomento, la funzione POINT ritorna un intero che rappresenta il colore nell'indirizzo specificato. In alternativa, la funzione POINT ritorna la coordinata orizzontale o verticale dell'ultimo punto a cui si è fatto riferimento sullo schermo grafico corrente. In questo caso, l'argomento intero seleziona una delle quattro opzioni di conversione:

1 Ritorna la coordinata *x* dello schermo.

2 Ritorna la coordinata *y* dello schermo.

3 Ritorna la coordinata *x* dell'area grafica.

4 Ritorna la coordinata *y* dell'area grafica.

Se si stanno convertendo le coordinate che sono all'interno dell'area grafica, si deve adattare il risultato di POINT con un appropriato spiazzamento VIEW per trovare gli indirizzi equivalenti dello schermo.

Un esempio di POINT

Il programma seguente è progettato per dimostrare la seconda sintassi della funzione POINT. (Si può trovare un esercizio simile nella voce PMAP.) Il programma stabilisce due aree grafiche nel modo schermo SCREEN 9, disegna un cerchio all'interno di ogni area grafica ed utilizza POINT per trovare gli indirizzi equivalenti dello schermo del centro del cerchio. Poi il programma ripristina l'area grafica di default e il sistema di coordinate e disegna una linea dal centro di un cerchio al centro di un altro. Gli indirizzi dello schermo di questa linea sono le coordinate che sono state calcolate con la funzione POINT:

```
' dimostrazione della funzione POINT.

DEFINT X-Y

SCREEN 9
COLOR 4, 15

VIEW (10, 10)-(210, 210), 4, 4
WINDOW (-10, -10)-(10, 10)
PSET(0, 0), 15
x1 = 10 + POINT(0)
y1 = 10 + POINT(1)
CIRCLE (0, 0), 8, 15

VIEW (250, 20)-(450, 320), 5, 5
WINDOW (-10, -10)-(10, 10)
PSET(0, 0), 15
x2 = 250 + POINT(0)
y2 = 20 + POINT(1)
```

```
CIRCLE (0, 0), 9, 15

VIEW
WINDOW
LINE (x1, y1)-(x2, y2), 15
```

I centri dei due cerchi sono rappresentati dagli indirizzi dello schermo *(x1, y1)* e *(x2, y2)*. Si noti che il programma aggiunge uno spiazzamento al valore di POINT per calcolare questi indirizzi. In ogni caso lo spiazzamento è preso dall'indirizzo dello schermo nell'angolo in alto a sinistra dell'area grafica corrente:

```
x1 = 10 + POINT(0)
y1 = 10 + POINT(1)

x2 = 250 + POINT(0)
x2 = 20 + POINT(1)
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione POINT.

LA FUNZIONE SCREEN

La funzione SCREEN ritorna i colori attuali o il valore ASCII del carattere visualizzato in una data posizione di riga o colonna sullo schermo.

```
SCREEN (rig, colon)          ' Dà il valore ASCII del
                             ' carattere in rig, colon.

SCREEN (rig, colon, col)     ' Se col è vero, dà i
                             ' colori in rig, colon.
```

Gli argomenti *rig* e *colon* della funzione SCREEN identificano la posizione di un carattere sullo schermo. Per default, SCREEN ritorna un intero che va da 0 a 255 e che rappresenta il valore ASCII visualizzato sullo schermo in una posizione specificata. Se si fornisce un valore logico vero per il terzo argomento opzionale, SCREEN ritorna un intero che rappresenta i colori in *rig, colon*. Nel modo SCREEN 0 si possono usare le seguenti formule per convertire questo valore di ritorno nei colori dello sfondo e del primo piano:

```
primo piano = SCREEN(rig, colon, TRUE) MOD 16
sfondo = ((SCREEN(rig, colon, TRUE) - sfondo) / 16) MOD 128
```

Un esempio della funzione SCREEN

Il seguente programma è progettato come dimostrazione della funzione SCREEN. Il programma riempie le prime 20 righe dello schermo di solo testo (modo SCREEN 0) con caratteri selezionati a caso in colori casuali. Poi il programma usa SCREEN per identificare il carattere ed i colori di sfondo e di primo piano in una posizione nel mezzo di questo gruppo di caratteri:

```
' dimostrazione della funzione SCREEN.

RANDOMIZE TIMER
DEF FNRand% (alt%, bas%) = bas% + INT(RND * (hi% - bas% + 1))

DIM col$(16)
FOR i% = 0 TO 15
   READ col$(i%)
NEXT i%

SCREEN 0
CLS
FOR i% = 1 TO 1600
   COLOR FNRand%(0, 16), FNRand%(0, 16)
   PRINT CHR$(FNRand%(33, 90));
NEXT i%

COLOR 7, 0
LOCATE 21, 15
PRINT "Il carattere nella riga 10, colonna 40 è ";
PRINT CHR$(SCREEN(10, 40)); "."

pripia% = SCREEN(10, 40, 1) MOD 16
sfond% = ((SCREEN(10, 40, 1) - pripia%) / 16) MOD 128

LOCATE 22, 15
PRINT "Il colore di primo piano di questo carattere è ";
PRINT col$(pripia%); "."
LOCATE 23, 15
PRINT "Il colore dello sfondo di questo carattere è ";
PRINT col$(sfond%); "."

END
```

```
DATA nero, blu, verde, ciano, rosso, magenta, marrone, bianco
DATA grigio, blu chiaro, verde chiaro, ciano chiaro, rosso chiaro
DATA magenta chiaro, giallo, bianco ad alta intensità
```

Il programma visualizza sullo schermo tre linee di messaggi, proprio sotto le 20 righe di caratteri casuali. Ecco un esempio di questi messaggi:

```
Il carattere alla riga 10, colonna 40 è A.
Il colore di primo piano di questo carattere è giallo.
Il colore di sfondo di questo carattere è rosso.
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione SCREEN.

L'ISTRUZIONE SCREEN

L'istruzione SCREEN seleziona ed attiva un particolare modo testo o schermo grafico.

```
SCREEN mod, col, pag1, pag2  ' Ogni argomento è opzionale: mod
                             ' seleziona il modo schermo; col
                             ' attiva o disattiva il colore; pag1
                             ' seleziona l'output di una pagina;
                             ' pag2 seleziona la pagina visualizzata.
```

L'argomento *mod* è un intero, che specifica un modo schermo che va da 0 a 4 o da 7 a 13. (I numeri di modo 5 e 6 non vengono usati abitualmente.) Il numero di modi con cui si lavora dipende dal tipo di hardware grafico che è installato nel proprio computer. Gli schermi IBM standard comprendono l'Adattatore Grafico a Colori (CGA), l'Adattatore Grafico Avanzato (EGA), VGA e MCGA. Inoltre, l'istruzione SCREEN supporta i modi per molti altri video. L'argomento opzionale *col* è un valore logico che si riferisce solo ai modi 0 e 1. Commuta tra uno schermo a colori e uno in bianco e nero. Nel modo 0 un valore logico vero dà immagini a colori ed un valore falso dà immagini in bianco e nero. Gli argomenti opzionali *pag1* e *pag2* si riferiscono alle "pagine" della memoria del video, ognuna delle quali memorizza una sola videata di informazioni grafiche. L'argomento *pag1* è

un intero (con un valore minimo pari a 0) che specifica la pagina a cui le istruzioni grafiche invieranno l'output. L'argomento *pag2* è la pagina che viene correntemente visualizzata sullo schermo. Usando queste opzioni, il programma può essere impegnato a preparare una videata mentre un'altra è già visualizzata. Si usa *pag2* per passare da una pagina visualizzata all'altra. Il numero di pagine disponibili dipende dal modo schermo e dalla quantità di memoria disponibile nel video, come descritto qui sotto. (Vedere anche la voce PCOPY per un esempio di grafici a più pagine.) La risoluzione in pixel varia da modo a modo. Per default, il pixel situato nell'angolo in alto a sinistra di ogni schermo ha l'indirizzo (0, 0). Comunque, si possono usare i comandi VIEW e/o WINDOW per stabilire altri sistemi di coordinate per l'intero schermo o per ogni porzione rettangolare dello schermo. Le istruzioni COLOR e PALETTE controllano l'uso del colore nei vari modi dello schermo. Queste istruzioni lavorano differentemente in diversi modi. In generale, si deve fare una distinzione tra gli *attributi* ed i corrispondenti *colori* che sono loro assegnati. La serie di interi degli attributi per un dato modo determina il numero di colori che possono comparire contemporaneamente sullo schermo. Il colore attuale rappresentato da un particolare numero di attributo può cambiare durante il corso di un programma. L'istruzione PALETTE assegna specifici colori del video agli attributi. Alcuni modi hanno disponibili molti più colori che attributi. La parte seguente descrive i modi dello schermo che vengono controllati dall'istruzione SCREEN. Ogni parte cita i video per i quali il modo è disponibile, la risoluzione in pixel, i colori e gli attributi, e dove possibile le dimensioni delle pagine.

SCREEN 0

Disponibilità: adattatori monocromatici, CGA, EGA, VGA.
Risoluzione grafica: nessuna.
Testo: 80 colonne per 25 righe sugli adattatori monocromatici; 80 per 25, o 40 per 25 su CGA; modi testo a 43 e 50 righe disponibili su EGA e VGA.
Colori ed attributi: 16 colori per 16 attributi su un sistema CGA o 64 colori assegnati a 16 attributi sui sistemi VGA e EGA.
Commenti: SCREEN 0 visualizza solo il testo ed è il modo di default. Il comando WIDTH controlla il numero di colonne e righe del testo.

SCREEN 1

Disponibilità: CGA, EGA, VGA, MCGA.
Risoluzione grafica: 320 pixel orizzontali per 200 pixel verticali con indirizzi di default (0, X0) a (319, X199).
Testo: 40 colonne per 25 righe.
Colori ed attributi: due tavolozze con 3 colori di primo piano ciascuna e 16 colori di sfondo su un sistema CGA o 16 colori assegnati a 4 attributi sui sistemi EGA e VGA.
Commenti: SCREEN 1 è disponibile su tutti gli schermi grafici standard IBM. Su un sistema CGA le due serie di colori sono la tavolozza 0 e la tavolozza 1; i singoli colori in queste tavolozze sono numerati 1, 2 e 3 come segue:

tavolozza 0: verde (1), rosso (2) e marrone (3)

tavolozza 1: ciano (1), magenta (2) e bianco fosforescente (3)

SCREEN 2

Disponibilità: CGA, EGA, VGA, MCGA.
Risoluzione grafica: 640 pixel orizzontali per 200 pixel verticali con indirizzi di default (0, 0) a (639, 199).
Testo: 80 colonne per 25 righe.
Colori ed attributi: un solo colore su un sistema CGA o 16 colori assegnati a 2 attributi sui sistemi EGA e VGA.
Commenti: non si deve usare l'istruzione COLOR in questo modo. L'istruzione PALETTE assegna i colori ai due attributi sui due sistemi EGA e VGA. I colori di default sono nero e bianco ad alta intensità.

SCREEN 3

Disponibilità: adattatore video Hercules.
Risoluzione grafica: 720 pixel orizzontali per 348 pixel verticali con indirizzi di default da (0, 0) a (719, 347).

Testo: 80 colonne di caratteri per 25 righe.
Colori ed attributi: un solo colore.
Commenti: si deve attivare il driver dell'adattatore Hercules di QuickBASIC (MSHERC.COM) prima di cercare di compilare i programmi in SCREEN 3. Bisogna immettere il seguente comando dal prompt del DOS prima di iniziare una parte nell'ambiente QuickBASIC:

```
C>MSHERC
```

Le istruzioni COLOR e PALETTE non sono disponibili.

SCREEN 4

Disponibilità: personal computer AT&T e Olivetti.
Risoluzione grafica: 640 pixel orizzontali per 400 pixel verticali con indirizzi di default da (0, 0) a (639, 399).
Testo: 80 colonne di caratteri per 25 righe.
Colori ed attributi: 16 colori di primo piano su uno sfondo nero.
Commenti: in questo modo l'istruzione PALETTE non è disponibile.

SCREEN 7

Disponibilità: EGA, VGA.
Risoluzione grafica: 320 pixel orizzontali per 200 pixel verticali con indirizzi di default da (0, 0) a (319, 199).
Testo: 40 colonne di caratteri per 25 righe.
Colori ed attributi: 16 colori per 16 attributi.
Commenti: il numero di pagine dipende dalla memoria disponibile sulla scheda dell'adattatore video; 32Kb richiesti per pagina.

SCREEN 8

Disponibilità: EGA, VGA.
Risoluzione grafica: 640 pixel orizzontali per 200 pixel verticali con indirizzi di default da (0, 0) a (639, 199).

Testo: 80 colonne di caratteri per 25 righe.
Colori ed attributi: 16 colori per 16 attributi.
Commenti: il numero di pagine dipende dalla memoria disponibile sulla scheda dell'adattatore video; 64Kb richiesti per pagina.

SCREEN 9

Disponibilità: EGA, VGA.
Risoluzione grafica: 640 pixel orizzontali per 350 pixel verticali con indirizzi di default da (0, 0) a (639, 349).
Testo: 80 colonne di caratteri per 25 o 43 righe.
Colori ed attributi: dipendono dalla memoria disponibile sulla scheda dell'adattatore, 64 colori assegnati a 16 attributi oppure 16 colori a 4 attributi.
Commenti: su uno schermo a colori a 4 attributi, le dimensioni della pagina sono di 64Kb e su uno schermo a 16 attributi le dimensioni sono di 128Kb. Il numero di pagine dipende dalla memoria disponibile sulla scheda dell'adattatore video.

SCREEN 10

Disponibilità: EGA, VGA.
Risoluzione grafica: 640 pixel orizzontali per 350 pixel verticali con indirizzi di default da (0, 0) a (639, 349).
Testo: 80 colonne di caratteri per 25 o 43 righe.
Colori ed attributi: solo un colore.
Commenti: le dimensioni attuali della pagina sono di 64Kb, ma sono richiesti 128Kb di memoria del video per ogni pagina. Nove valori "colore" assegnati a 4 attributi rappresentano le combinazioni del testo ad intensità normale, alta e lampeggiante.

SCREEN 11

Disponibilità: EGA, MCGA.

Risoluzione grafica: 640 pixel orizzontali per 480 pixel verticali con indirizzi di default da (0, 0) a (639, 479).
Testo: 80 colonne di caratteri per 30 o 60 righe.
Colori ed attributi: qualsiasi colore fra i 262.144 assegnati a 2 attributi.
Commenti: gli interi lunghi che rappresentano i colori dello schermo VGA vengono calcolati con questa formula:

```
col = (b * (256 ^ 2)) + (v * 256) + r
```

dove *b*, *v* e *r* sono interi che vanno da 0 a 63 e che rappresentano le intensità e le sfumature rispettivamente del blu, verde e rosso. (Si noti che questa formula *non* compare in un intervallo contiguo di interi che va da 1 a 262.144. Piuttosto l'intervallo degli interi del colore va da 0 a più di 4 milioni; ma solo 262.144 interi di questo intervallo rappresentano effettivamente i colori.) L'istruzione COLOR non è disponibile nel modo 11; si usa l'istruzione PALETTE per selezionare i colori per i due attributi. Le dimensioni delle pagine nel modo 11 sono di 64Kb; è disponibile una pagina.

SCREEN 12

Disponibilità: VGA.
Risoluzione grafica: 640 pixel orizzontali per 480 pixel verticali con indirizzi di default da (0, 0) a (639, 479).
Testo: 80 colonne di caratteri per 30 o 60 righe.
Colori ed attributi: qualsiasi colore fra i 262.144 assegnati a 16 attributi.
Commenti: gli interi lunghi che rappresentano i colori dello schermo VGA vengono calcolati con questa formula:

```
col = (b * (256 ^ 2)) + (v * 256) + r
```

dove *b*, *v* e *r* sono interi che vanno da 0 a 63 e che rappresentano le intensità e le sfumature rispettivamente del blu, verde e rosso. Le dimensioni della pagina nel modo 12 sono di 256Kb; è disponibile una pagina.

SCREEN 13

Disponibilità: VGA, MCGA.
Risoluzione grafica: 320 pixel orizzontali per 200 pixel verticali con indirizzi di default da (0, 0) a (319, 199).
Testo: 40 colonne di caratteri per 25 righe.
Colori ed attributi: qualsiasi colore fra i 262.144 assegnati a 256 attributi.
Commenti: gli interi lunghi che rappresentano i colori dello schermo VGA vengono calcolati con questa formula:

```
col = (b * (256 ^ 2)) + (v * 256) + r
```

dove *b*, *v* e *r* sono interi che vanno da 0 a 63 e che rappresentano le intensità e le sfumature rispettivamente del blu, verde e rosso. Le dimensioni della pagina nel modo 13 sono di 64Kb; è disponibile una pagina.

Un esempio di SCREEN

Il programma *DateStoriche*, listato nel Capitolo 32, opera nel modo grafico 9:

```
SCREEN 9
```

Questo modo è disponibile solo sui sistemi EGA e VGA e ha una risoluzione a pixel di 640 pixel orizzontali per 350 pixel verticali, con indirizzi di default da (0, 0) a (639, 349). Il programma ripristina attentamente il modo testo prima di terminare:

```
SCREEN 0
```

Questo è un passo finale importante in un'applicazione grafica, specialmente quando si anticipa che il programma verrà attivato dal prompt del DOS come un file EXE compilato a funzionamento autonomo. Questo strumento del programma esegue una serie di test su uno schermo dato e produce una lista di tutti i modi SCREEN disponibili nel sistema. Il programma *TestModo* usa una gestione degli errori per determinare se un dato modo è valido per il sistema corrente:

```
' MODETEST.BAS
' Determina quali modi SCREEN sono disponibili sullo
' schermo corrente.

CONST FALSE = 0
CONST TRUE = NOT FALSE
DIM mod%(13)

' Tenta tutti i modi grafici.

FOR i% = 1 TO 13
   modOk% = TRUE
   ON ERROR GOTO ModErr
      SCREEN i%
      PSET (0, 0)
   ON ERROR GOTO 0
   mod%(i%) = modOk%
NEXT i%

' Visualizza i modi possibili.

SCREEN 0: WIDTH 80
PRINT "Ecco i modi SCREEN grafici"
PRINT "ammessi su questo schermo:"
PRINT
FOR i% = 1 TO 13
   IF mod%(i%) THEN PRINT " SCREEN"; i%
NEXT i%

END

' Routine di gestione degli errori chiamata
' se un modo non è ammesso.

ModErr:
   modOk% = FALSE
RESUME NEXT
```

Il programma tenta di commutare ogni modo schermo grafico, uno alla volta, e rappresenta graficamente sullo schermo ogni modo. Una gestione degli errori passa il controllo ad una routine di gestione degli errori chiamata *ModErr* se una data istruzione SCREEN risulta in un errore run-time. Questa routine passa un valore logico falso indietro alla variabile *modOk%*, che indica che il modo non è valido. Si può usare una tecnica simile per assicurarsi che un data applicazione grafica sarà utilizzabile in qualsiasi videata grafica; il programma può semplicemente far passare i modi grafici fin che non trova un modo che non dà un errore. Dopo aver esaminato tutti

i modi, il programma *TestModo* visualizza una lista dei modi che si possono usare nel sistema corrente. Ad esempio, ecco la lista presentata su un video a più modi:

```
Ecco i modi grafici SCREEN ammessi su questo schermo:

SCREEN 1
SCREEN 2
SCREEN 7
SCREEN 8
SCREEN 9
SCREEN 10
SCREEN 11
SCREEN 12
SCREEN 13
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione SCREEN. Supportare vari modi dipende generalmente dall'hardware del video che era disponibile quando è stata sviluppata una data versione del linguaggio.

VIEW

L'istruzione VIEW definisce un'*area grafica* all'interno dei comandi grafici che si usano.

```
VIEW (x1, y1)-(x2, y2)          ' L'angolo in alto a sinistra del
                                ' sistema di indirizzamento
                                ' dell'area grafica diventa (0, 0).


VIEW SCREEN (x1, y1)-(x2, y2)   ' (x1, y1) e (x2, y2) sono gli angoli
                                ' del sistema di indirizzamento
                                ' dell'area grafica.


VIEW (x1, y1)-(x2, y2), col, con   ' col e con sono opzionali.
VIEW                               ' L'intero schermo è ristabilito
                                   ' come area grafica.
```

L'istruzione VIEW dà l'opportunità di limitare l'attività grafica ad una porzione definita dello schermo. Usata con l'istruzione WINDOW (che definisce un nuovo sistema di coordinate per l'area grafica corrente), VIEW risulta nelle adeguate tecniche di indirizzamento per una particolare applicazione grafica. Gli indirizzi dello schermo *(x1, y1)* e *(x2, y2)* rappresentano gli angoli opposti dell'area grafica che è definita dall'istruzione VIEW. Per default, l'angolo in alto a sinistra dell'area grafica ha un nuovo indirizzo di (0, 0) e l'angolo in basso a destra diventa *(ABS(x2-x1), ABS(y2-y1))*. Dopo l'istruzione VIEW, ogni figura che il programma cerca di visualizzare sullo schermo deve essere posizionata all'interno di questi indirizzi altrimenti non compare. In alternativa, si può includere la parola chiave opzionale SCREEN nell'istruzione VIEW. In questo caso, il sistema d'indirizzo all'interno dell'area grafica rimane immutato: gli angoli opposti dell'area grafica sono ancora rappresentati dagli indirizzi attuali *(x1, y1)* e *(x2, y2)*. Ciononostante, l'attività grafica è ancora limitata all'area grafica. Ci sono due opzioni aggiuntive nell'istruzione VIEW. L'argomento *col* specifica un colore di sfondo per l'intera area grafica e l'argomento *con* è il colore del perimetro dell'area grafica. Questi due colori non devono essere uguali. Infine, l'istruzione VIEW da sola, senza argomenti, ripristina l'intera area dello schermo come area grafica.

Alcuni esempi di VIEW

Il programma *DateStoriche* (presentato nel Capitolo 32) usa le istruzioni VIEW e WINDOW per stabilire un'area grafica per l'orologio che il programma visualizza nell'angolo in basso a destra dello schermo. Per prima cosa l'istruzione VIEW definisce l'area grafica:

```
VIEW (399, 174)-(639, 349)
```

Poi un'istruzione WINDOW crea un semplice sistema di coordinate per l'area grafica:

```
WINDOW (-10, -10)-(10, 10)
```

Le istruzioni grafiche che successivamente inseriscono le figure in questa area grafica hanno riferimenti di indirizzo molto semplici. Ad esempio, i cerchi che costituiscono l'orologio hanno come indirizzi del centro (0, 0):

```
CIRCLE (0, 0), lunghr% - 1, 4
PAINT (0, 0), 15, 4
CIRCLE (0, 0), .25, 1
PAINT (0, 0), 1, 1
```

Inoltre, gli intervalli della lancetta dei secondi dell'orologio vanno da (0, 0) ad un punto nella circonferenza esterna dell'orologio:

```
LINE (0, 0)-(0, lunghr%), 4
```

Quando altre parti del programma devono inserire figure nell'area esterna a quella grafica, le seguenti istruzioni ripristinano l'intero schermo nel suo originale sistema di indirizzamento:

```
VIEW
WINDOW
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione VIEW.

WIDTH

L'istruzione WIDTH controlla il numero di colonne e righe del testo visualizzato sullo schermo. WIDTH può anche essere usato per controllare la larghezza della linea per l'output ad altri dispositivi.

```
WIDTH col, rig       ' Stabilisce il numero di colonne di caratteri
                     ' e righe sullo schermo.

WIDTH LPRINT col     ' Stabilisce il numero di colonne per linea
                     ' sulla stampante.

WIDTH #num, col      ' Stabilisce lo spessore della linea per un file
                     ' aperto o un dispositivo d'output.

WIDTH "disp", col    ' Stabilisce lo spessore della linea per
                     ' un dispositivo d'output.
```

Per controllare la larghezza del testo visualizzato sullo schermo, l'argomento *col* è il valore intero 80 o 40 e l'argomento *rig* sceglie fra le opzioni dell'ampiezza che sono 25, 30, 43, 50 o 60. Le specifiche opzioni disponibili dipendono dal modo video. (Per ulteriori dettagli consultare la voce SCREEN.) L'istruzione WIDTH LPRINT controlla il numero di colonne per linea inviate alla stampante. L'argomento *#num* in un'istruzione WIDTH si riferisce ad un file che è aperto per l'output. In questa sintassi, l'istruzione WIDTH specifica lo spessore delle linee inviate al file. Infine, l'argomento *"disp"* è un valore stringa che identifica un particolare dispositivo, come "COM1:" o "COM2:". Questa istruzione ha effetto dopo che il dispositivo è stato aperto un'*altra* volta.

Un esempio di WIDTH

Il seguente programma dimostra l'uso di WIDTH per visualizzare il numero massimo di linee nel modo SCREEN 12. Il programma visualizza il testo su uno schermo a 80 colonne per 60 righe:

```
SCREEN 12
WIDTH 80, 60
FOR i% = 1 TO 60
   LOCATE i%, 20
   PRINT "Questa è la linea #"; i%;
NEXT i%
SLEEP
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione WIDTH. Comunque, in queste versioni del linguaggio la sintassi dell'istruzione si limita a controllare la larghezza in colonne dell'output allo schermo o ad un altro dispositivo. L'opzione *rig* non è supportata.

WINDOW

L'istruzione WINDOW definisce le coordinate dell'area grafica corrente.

```
WINDOW (x1, y1)-(x2, y2)           ' (x1, y1) e (x2, y2) definisce il
                                   ' nuovo sistema di coordinate.

WINDOW SCREEN (x1, y1)-(x2, y2)    ' Viene invertita la direzione
                                   ' delle coordinate y.

WINDOW                             ' Viene ripristinato il sistema
                                   ' originale di coordinate.
```

L'istruzione WINDOW dà l'opportunità di semplificare il sistema di indirizzamento per una particolare applicazione grafica. Nella maggior parte dei casi WINDOW viene eseguita dopo l'istruzione VIEW che stabilisce un'area grafica sullo schermo grafico corrente. WINDOW crea un opportuno sistema di coordinate all'interno di quest'area grafica. Gli indirizzi *(x1, y1) – (x2, y2)* nell'istruzione WINDOW rappresentano le nuove coordinate per gli angoli opposti dell'area grafica attuale. Per default, gli indirizzi stabiliti dall'istruzione WINDOW sono conformi alle convenzioni di un sistema di coordinate cartesiane: i valori *x* aumentano da sinistra verso destra ed i valori *y* dal basso verso l'alto dell'area grafica. Comunque, si può modificare la componente verticale di questo sistema inserendo la parola chiave opzionale SCREEN nell'istruzione WINDOW. Così facendo, la direzione dei valori *y* viene invertita: le coordinate *y* aumentano dall'alto verso il basso. Per ripristinare il sistema originale d'indirizzamento, bisogna usare l'istruzione WINDOW senza argomenti.

Alcuni esempi di WINDOW

Il programma *DateStoriche* (presentato nel Capitolo 32) usa WINDOW per ottenere un sistema adatto di coordinate per disegnare un orologio nell'angolo in basso a destra dello schermo. Per prima cosa l'istruzione VIEW crea l'appropriata area grafica e poi WINDOW stabilisce gli indirizzi delle coordinate di quest'area:

```
VIEW (399, 174)-(639, 349)
WINDOW (-10, -10)-(10, 10)
```

Le istruzioni grafiche che in seguito inseriscono le figure in quest'area contengono delle referenze d'indirizzo molto semplici. Ad esempio, il

cerchio che definisce l'orologio ha come indirizzi del centro (0, 0). WINDOW crea un semplice sistema di coordinate per l'area grafica:

```
CIRCLE (0, 0), lunghr% - 1, 4
```

Altre parti del programma ristabiliscono il sistema di indirizzamento di default inserendo questi due comandi:

```
VIEW
WINDOW
```

Per ulteriori dettagli vedere la voce VIEW.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione WINDOW.

Capitolo 16

Colore

Questo capitolo esamina tre importanti istruzioni che si possono usare per selezionare e visualizzare i colori nei modi testo e grafico dello schermo:

- L'istruzione COLOR seleziona i colori di sfondo e di primo piano per un dato modo.
- Le istruzioni PALETTE e PALETTE USING assegnano i colori dello schermo usando gli attributi disponibili di un dato modo.
- L'istruzione PAINT colora una figura chiusa sullo schermo.

L'utilizzo e la sintassi delle istruzioni di controllo del colore tendono a cambiare da un modo schermo all'altro. In generale, queste istruzioni si differenziano per gli *attributi* disponibili nel modo dato, cioè la serie di interi che rappresenta il numero di colori che possono comparire sul video contemporaneamente e gli stessi colori dello schermo. Questa distinzione va ricordata quando si usano queste ed altre istruzioni grafiche di QuickBASIC.

COLOR

L'istruzione COLOR seleziona i colori di sfondo e di primo piano per il modo schermo corrente.

```
COLOR primp, sfond, cont     ' SCREEN 0.
COLOR sfond, tavol           ' SCREEN 1.

COLOR primp, sfond           ' SCREEN mod 7 a 10.
COLOR primp                  ' SCREEN mod 12 e 13.
```

La sintassi e l'uso dell'istruzione COLOR cambiano in base al modo schermo corrente. Nella sintassi per un dato modo, dove appropriato, è possibile dare un valore solo per il primo piano o solo per lo sfondo o solo per il contorno. Per ogni argomento mancante occorre inserire una virgola come segnaposto. Nel modo SCREEN 0, l'argomento *primp* rappresenta il colore dello schermo per il testo. Questo argomento può essere un intero che va da 0 a 31, dove i primi 16 valori sono i seguenti:

0	Nero
1	Blu
2	Verde
3	Ciano
4	Rosso
5	Magenta
6	Marrone
7	Bianco
8	Grigio
9	Blu chiaro
10	Verde chiaro
11	Ciano chiaro
12	Rosso chiaro (o arancione)
13	Magenta chiaro
14	Giallo
15	Bianco ad alta intensità

Bisogna aggiungere 16 ad un codice del colore per ottenere che il testo di quel colore lampeggi. Il colore di sfondo è un intero che va da 0 a 7. Si può

scegliere un diverso sfondo per ogni carattere del testo. Sui video CGA il colore del contorno è un intero da 0 a 15. Gli altri adattatori video ignorano l'argomento *cont*. Nel modo SCREEN 1 su un sistema CGA, la selezione del colore si limita a due tavolozze prestabilite. L'istruzione COLOR sceglie una di queste tavolozze per i colori di primo piano ed un solo colore per lo sfondo. Le tavolozze di default sono numerate 0 e 1 e hanno tre colori ciascuna, numerati 1, 2 e 3 nel modo seguente:

tavolozza 0: verde (1), rosso (2) e marrone (3)

tavolozza 1: ciano (1), magenta (2) e bianco fosforescente (3)

Su altri video si può usare l'istruzione PALETTE per cambiare i numeri di codice corrispondenti ai vari colori all'interno delle due tavolozze; ma il numero di colori all'interno di ogni tavolozza rimane costante. Nei modi schermo da 7 a 10 (per i sistemi EGA e VGA) l'argomento *primp* nell'istruzione COLOR è un attributo compreso nell'intervallo degli attributi validi. L'istruzione PALETTE controlla il colore corrispondente per gli attributi. (Consultare la voce PALETTE per ulteriori dettagli.) Al contrario, l'argomento *sfond* è il numero del colore effettivo dello schermo, non un attributo. Il modo SCREEN 11 non utilizza l'istruzione COLOR. L'istruzione PALETTE controlla a quali colori corrispondono i due attributi di questo modo. Nei modi 12 e 13 (VGA e MCGA) si può usare l'istruzione COLOR per selezionare un colore di primo piano. L'argomento *primp* è un intero compreso nell'intervallo valido dei numeri di attributo. L'istruzione PALETTE assegna i colori dello schermo agli attributi. Si può controllare il colore di sfondo assegnando un colore all'attributo 0 con un'istruzione PALETTE.

Un esempio COLOR

Il programma *DateStoriche*, presentato nel Capitolo 32, seleziona i colori per SCREEN 9 così:

```
SCREEN 9
COLOR 1, 15
```

Questo dà un primo piano blu per il testo del programma, mentre lo sfondo è blu chiaro.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione COLOR.

PAINT

L'istruzione PAINT riempie un'area delimitata da contorni con un colore o un tratteggio.

```
PAINT (x, y), riem, cont, trat       ' riem, cont, e trat sono
                                     ' opzionali.

PAINT STEP (x, y), riem, cont, trat  ' usa un indirizzo relativo.
```

L'indirizzo nell'istruzione PAINT è un punto nell'area che deve essere colorata. Si può identificare questo punto come un indirizzo assoluto *(x, y)* o come un indirizzo relativo, STEP *(x, y)*. (Un indirizzo relativo viene espresso come una distanza orizzontale *x* e come una verticale *y*, dall'ultimo punto a cui si è fatto riferimento sullo schermo.) L'argomento *cont* è il numero dell'attributo che rappresenta il colore che racchiude l'area in questione. PAINT riempie l'interno o l'esterno dell'area, in base alla posizione del punto specificato. L'argomento *riem* può presentarsi o come un intero o come una stringa. Un valore intero è un attributo che rappresenta il colore con cui PAINT riempirà l'area. Se si omette *riem*, PAINT usa il colore corrente di primo piano come default. (Se si omette *cont*, il colore di *riem* viene preso come colore del contorno per default.) Se si assegna un valore all'argomento *riem*, PAINT converte la stringa in un tratteggio coprente che riempie l'area stabilita. In questo caso, ogni carattere ASCII nella stringa *riem* viene considerato come sequenza di cifre binarie, cioè viene considerato l'equivalente binario del numero di codice ASCII del carattere. Il valore binario 1 corrisponde a "attivato" e quello 0 a "disattivato". Questo individua uno specifico tratteggio a colori o in bianco e nero, dipende dal video. (Vedere la voce LINE per uno strumento che

fornisca l'intero decimale equivalente ad una stringa di cifre binarie.) La stringa *riem* può contenere fino a 64 caratteri. L'argomento finale facoltativo nell'istruzione PAINT, *trat*, è una seconda stringa per il tratteggio. Si usi questo argomento se si vuole sovrapporre un tratteggio all'altro. In effetti, l'istruzione PAINT, mentre riempie l'area stabilita con il modello *riem*, ignora la visualizzazione di un tratteggio esistente.

Alcuni esempi di PAINT

Il programma *DateStoriche* usa l'istruzione PAINT per differenziare sullo schermo aree diverse colorandole. Ad esempio, il programma disegna sullo schermo un orologio che ha il contorno rosso e l'area interna bianco fosforescente. Il centro dell'orologio è dato da un cerchio blu molto più piccolo:

```
CIRCLE (0, 0), lunghlanc% - 1, 4
PAINT (0, 0), 15, 4
CIRCLE (0, 0), .25, 1
PAINT (0, 0), 1, 1
```

Allo stesso modo, il programma riempie due finestre rettangolari dello schermo con il bianco, per cancellare il testo che è all'interno delle finestre. Ad esempio, ecco come il programma crea una finestra per visualizzare una qualsiasi domanda:

```
LINE (47, 63)-(582, 92), 4, B
PAINT (48, 64), 15, 4
LOCATE 6, 8
PRINT infData(numRich%).event
```

Infine, il seguente programma è una breve dimostrazione della tecnica di copertura dell'istruzione PAINT:

```
SCREEN 9
COLOR 1, 15

LINE (100, 100)-(550, 250), 4, B
PAINT (150, 150), "abcABC123", 4

SLEEP
SCREEN 0
```

Quando si attiva questo programma si vede che il contenuto del testo della stringa *riem* non ha niente a che fare con il tratteggio risultante. Infatti, PAINT usa l'equivalente binario del codice ASCII di ogni carattere per costruire il tratteggio coprente.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione PAINT.

PALETTE E PALETTE USING

Le istruzioni PALETTE e PALETTE USING assegnano nuovi colori video a uno o più attributi. Ciò risulta in un'immediata sostituzione dei colori attualmente visualizzati sullo schermo.

```
PALETTE attr, col            ' Cambia un attributo.

PALETTE USING vett(elem)     ' Assegna ad una serie di attributi
                             ' i numeri del  colore memorizzati in
                             ' vett. Il facoltativo argomento (elem)
                             ' indica il punto d'inizio nel vettore.

PALETTE                      ' Ripristina i colori di default in tutti
                             ' gli attributi nel modo schermo corrente.
```

Ogni modo schermo ha un valido intervallo di *attributi*, gli interi che rappresentano i colori nelle istruzioni per disegnare e per usare il plotter come PSET, LINE, CIRCLE e DRAW. Il numero di attributi in un modo dato corrisponde al numero di colori che possono apparire sullo schermo contemporaneamente. Ogni attributo corrisponde per default ad un colore. La serie di colori dello schermo corrispondente agli attributi disponibili talvolta è chiamata *tavolozza*. In molti modi il numero di colori dello schermo possibili è maggiore, anzi consistentemente maggiore, del numero degli attributi. Sui video EGA, VGA e MCGA si possono usare le istruzioni PALETTE o PALETTE USING per assegnare nuovi colori video agli attributi. Così facendo, i colori corrispondenti sullo schermo vengono immediatamente sostituiti dai colori nuovi assegnati. (Per ulteriori informazioni sugli attributi in specifici modi schermo, vedere l'istruzione SCREEN.) La prima sintassi

dell'istruzione PALETTE assegna un nuovo colore ad un solo attributo. L'argomento *attr* è un intero compreso nell'intervallo valido di attributi nel modo schermo corrente. L'argomento *col* è un intero o un intero lungo che rappresenta un colore video disponibile. Ogni colore video è rappresentato, in un modo dato, da un numero fisso. Ad esempio, gli interi da 0 a 15 rappresentano i seguenti colori nella maggior parte dei modi:

0	Nero
1	Blu
2	Verde
3	Ciano
4	Rosso
5	Magenta
6	Marrone
7	Bianco
8	Grigio
9	Blu chiaro
10	Verde chiaro
11	Ciano chiaro
12	Rosso chiaro (o arancione)
13	Magenta chiaro
14	Giallo
15	Bianco ad alta intensità

Questi colori sono anche gli assegnamenti di default per gli attributi che vanno da 0 a 15 nella maggior parte dei modi.
I modi SCREEN 11, 12 e 13 (per i video VGA e MCGA) hanno un vasto numero di colori disponibili, in tutto 262.144. Gli interi che rappresentano questi colori non formano però un intervallo contiguo di interi; il numero dei vari colori è calcolato in base alla seguente formula:

```
col = (b * (256 ^ 2)) + (v * 256) + r
```

dove *b, v* e *r* sono interi che vanno da 0 a 63 e che rappresentano le intensità e le sfumature rispettivamente di blu, verde e rosso. (Si noti che questa formula *non* risulta in un intervallo contiguo di interi che va da 1 a 262.144. L'intervallo degli interi del colore va invece da 0 a più di 4 milioni, anche se solo 262.144 interi di questo intervallo rappresentano effettivamente i colori.) Nell'istruzione PALETTE per questi modi, il numero del colore deve sempre essere espresso come un intero lungo. L'istruzione PALETTE USING assegna i colori ad una serie di attributi per volta. In questo caso, bisogna anticipatamente memorizzare i numeri dei colori video in un vettore definito per questo scopo. Il nome di questo vettore appare poi come argomento dell'istruzione PALETTE. Per default QuickBASIC assegna il codice colore memorizzato nell'elemento 0 del vettore all'attributo 0, l'elemento 1 all'attributo 1 e così via fino al numero di elementi nel vettore o al numero di attributi nel modo corrente, qualunque dei due venga per primo. Si può, comunque, mettere un numero opzionale *(elem)* dopo il nome del vettore; questo numero indica il punto d'inizio nel vettore da cui partono gli assegnamenti degli attributi. In questo modo, un solo vettore può contenere gli assegnamenti degli attributi per molti diversi cambiamenti di tavolozza. (Il programma presentato nel paragrafo "Esempio di PALETTE USING" illustra questa tecnica.) L'istruzione PALETTE sola, senza argomenti, ripristina i colori di default in tutti gli attributi.

Esempio di PALETTE USING

Il seguente programma disegna un cerchio di 15 settori concentrici su uno sfondo nero in SCREEN 9 e riempie ogni settore con un colore da 1 a 15. Poi il programma usa l'istruzione PALETTE USING all'interno di un ciclo FOR per cambiare più volte la tavolozza. Il programma definisce un vettore intero, chiamato *colUg%*, che contiene i codici numerici dei colori video da 0 a 63. Ogni esecuzione dell'istruzione PALETTE USING esegue degli assegnamenti di colore usando i codici contenuti in questo vettore da un diverso punto di partenza:

```
' Dimostrazione di PALETTE USING.

DEF FNrad (ang) = 8 * ATN(1) * (ang / 360)
```

```
SCREEN 9
COLOR , 0
WINDOW (219, 174)-(-219, -174)

DIM colUg%(63)
FOR i% = 1 TO 63
   colUg%(i%) = i%
NEXT i%

FOR i% = 0 TO 359 STEP 24
   CIRCLE (0, 0), 150, 4, -FNrad(i%), -FNrad(i% + 24)
   x = 100 * COS(FNrad(i% + 12))
   y = 100 * SIN(FNrad(i% + 12))
   PAINT (x, y), i% / 24, 4
NEXT i%

x = 100 * COS(FNrad(372))
y = 100 * SIN(FNrad(372))
PAINT (x, y), 15, 4

SLEEP

FOR p% = 0 TO 3
   PALETTE USING colUg%(p% * 16)
   SLEEP
NEXT p%

SCREEN 0
```

Durante l'esecuzione di questo programma, si può premere qualsiasi tasto della tastiera per visualizzare sullo schermo la successiva tavolozza di colori.

Compatibilità

Turbo Basic e le recenti versioni di BASIC Microsoft supportano l'istruzione PALETTE; questa non è disponibile per i video CGA.

Capitolo 17

Punti, linee e figure

Questo capitolo esamina i comandi di QuickBASIC che si usano per disegnare sullo schermo diverse figure geometriche, compresi punti, linee, rettangoli, cerchi, ellissi, archi, cunei e figure irregolari di ogni tipo. Le istruzioni CIRCLE, LINE, PSET e DRAW sono gli strumenti principali per creare immagini grafiche sul video. Questi comandi consentono l'uso di due tipi di indirizzi dello schermo: *assoluto* e *relativo*. Un indirizzo assoluto è costituito da una coppia ordinata di interi, *(x, y)*, dove *x* è l'indirizzo di un pixel orizzontale e *y* è l'indirizzo di un pixel verticale. Al contrario, un indirizzo relativo rappresenta un movimento in una specifica direzione partendo dall'ultimo punto a cui si era fatto riferimento sullo schermo. Un indirizzo relativo segue questa sintassi, usando la parola chiave STEP:

```
STEP (x, y)
```

In questo caso, *x* è un intero che rappresenta un movimento orizzontale partendo dall'ultimo punto a cui si è fatto riferimento. In generale, un valore negativo comporta un movimento verso sinistra mentre un valore positivo implica un movimento verso destra. Allo stesso modo, *y* è un intero che rappresenta un movimento verticale; per default un valore negativo designa uno spostamento verso l'alto ed uno positivo verso il basso.
Ad esempio, se l'ultimo punto a cui si era fatto riferimento era (25, 50), allora il seguente indirizzo relativo si muove verso (15, 60).

```
STEP (-10, 10)
```

Quando si passa per la prima volta ad un modo grafico, l'iniziale "ultimo punto a cui si è fatto riferimento" è il punto centrale dello schermo, ma qualsiasi nuova figura grafica che si disegnerà in seguito cambierà il valore dell'ultimo punto. Oltre i comandi per disegnare delle figure, QuickBASIC offre due importanti strumenti che eseguono veloci operazioni grafiche. GET prende una figura esistente dallo schermo e la memorizza in un vettore numerico. PUT utilizza il vettore per ridisegnare, molto velocemente, la figura in una posizione qualsiasi. Insieme, GET e PUT sono elementi essenziali per tutti i programmi che creano un'animazione. Molti degli esempi discussi in questo capitolo sono presi dal programma *DateStoriche*, presentato nel Capitolo 32. Inoltre, si possono trovare molti programmi interessanti ed utili a sé stanti, tra questi:

- Un programma chiamato *GetPut*, che descrive le istruzioni GET e PUT. Questo programma (listato nelle voci GET e PUT) contiene anche una procedura che calcola correttamente lo spazio di memoria richiesto per un vettore GET.
- Un programma chiamato *Disegna*, listato nella voce PSET. Questo programma consente di disegnare una figura sullo schermo interattivamente, usando i comandi della tastiera. Così facendo, il programma converte le operazioni in comandi stringa DRAW e li memorizza sul disco in un file di testo.

Questo ed altri programmi di questo capitolo aiuteranno i lettori ad usare e capire le istruzioni di QuickBASIC che disegnano delle figure sullo schermo grafico.

CIRCLE

L'istruzione CIRCLE disegna un cerchio o una figura simile su uno schermo grafico qualsiasi.

```
CIRCLE (x, y), lungh, col     ' Disegna un cerchio con raggio lungh
                              ' nel centro (x, y). L'argomento col
                              ' è opzionale.
```

```
CIRCLE STEP (x, y), lungh, col         ' Calcola il centro come
                                       ' un indirizzo relativo.

CIRCLE (x, y), lungh, col, ang1, ang2   ' Disegna un arco, dall'ang1
                                       ' all'ang2.

CIRCLE (x, y), lungh, col, -ang1, -ang2 ' Disegna un cuneo dall'ang1
                                       ' all'ang2.

CIRCLE (x, y), lungh, col, , , rap     ' Disegna un'ellisse con
                                       ' aspetto rap.
```

La figura prodotta dall'istruzione CIRCLE viene sempre visualizzata attorno al centro *(x, y)*. Si può esprimere questo punto come un indirizzo assoluto o si può usare la parola chiave STEP per indicare un indirizzo relativo, un movimento in avanti o indietro dall'ultimo punto a cui si è fatto riferimento sullo schermo. Dopo l'indirizzo, l'altro argomento richiesto nella sintassi dell'istruzione CIRCLE è *lungh*, il raggio del cerchio in pixel. L'argomento opzionale *col* identifica il colore con cui il cerchio verrà disegnato; se si omette questa opzione, il cerchio appare nel colore corrente di primo piano. Sono presenti ulteriori opzioni CIRCLE per modificare la figura che crea CIRCLE. Prima, gli argomenti *ang1* e *ang2* sono gli angoli iniziale e finale di un arco. Questi angoli sono espressi in radianti, da 0 a 2π. Se si inseriscono questi argomenti, CIRCLE disegna un arco, cioè una porzione di un cerchio, da *ang1* a *ang2*. Inoltre, se si assegnano valori negativi a questi due angoli, l'istruzione CIRCLE disegna un cuneo, cioè un arco con raggi laterali. Infine, si può usare l'opzione *rap* per disegnare una ellisse. In generale, una valore in frazione (minore di 1) genera un'ellisse orizzontale e una frazione maggiore di 1 dà un'ellisse verticale. Se si omette il valore *rap*, l'istruzione CIRCLE disegna un cerchio. Per agire in questo modo, QuickBASIC automaticamente calcola un rapporto appropriato per il modo schermo corrente. Questo conteggio si basa sul rapporto tra il numero di pixel orizzontali e verticali nel modo schermo attuale:

```
rap = (4 / 3) * (pixVert / pixOriz)
```

Alcuni esempi di CIRCLE

Il programma *DateStoriche*, presentato nel Capitolo 32, usa l'istruzione CIRCLE per disegnare un orologio nell'angolo in basso a destra del video:

```
CIRCLE (0, 0), lungh% -1, 4
PAINT (0, 0), 15, 4
CIRCLE (0, 0), .25, 1
PAINT (0, 0), 1, 1
```

Il cerchio grande rappresenta la circonferenza dell'orologio e quello più piccolo è il centro dell'orologio. Si noti che un'istruzione PAINT riempie con il colore l'interno di ogni cerchio. Questo breve programma dimostra le opzioni aggiuntive dell'istruzione CIRCLE. Il programma disegna un'ellisse orizzontale costituita da stretti cunei concentrici:

```
DEF FNrad (ang) = 8 * ATN(1) * (ang / 360)
SCREEN 9
COLOR 1, 15

FOR i% = 1 TO 360 STEP 8
   CIRCLE STEP(0, 0), 200, 4, -FNrad(i%), -FNrad(i% + 4), .375
NEXT i%

SLEEP
SCREEN 0
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione CIRCLE.

DRAW

Il comando DRAW disegna una figura sullo schermo grafico.

```
DRAW str
```

L'argomento stringa nell'istruzione DRAW contiene una qualsiasi combinazione di comandi in lettere che danno le istruzioni per disegnare:

- Le lettere U, D, L e R rappresentano le direzioni orizzontali e verticali di disegno, verso l'alto, il basso, destra e sinistra. Inoltre, anche le lettere che

seguono indicano le direzioni orizzontali: E (in alto a destra), F (in basso a destra), G (in basso a sinistra) e H (in alto a sinistra). Ognuna di queste lettere è solitamente seguita da un intero che esprime la lunghezza del tratto di linea. La lettera B dà uno spazio. La lettera N disegna la linea e poi rimanda alla posizione originale sullo schermo. Il comando M*x,y* sposta ad un nuovo indirizzo sul video, senza disegnare alcun grafico. (Più o meno simboli davanti a *x* e *y* danno un movimento relativo.)

- Il comando C*n* determina il colore del disegno, dove *n* rappresenta il colore del primo piano. Il comando P*n1,n2* riempie con il colore una figura chiusa; *n2* è il colore del contorno della figura e *n1* è il colore dell'interno.
- Il comando S*n* controlla la scala del disegno. La lunghezza scritta in comando grafico viene poi moltiplicata per *n*. Il default è S4. Il comando A*n* ruota il disegno per angoli multipli di 90 gradi. A1 è 90 gradi, A2 è 180 gradi e A3 è 270 gradi. Il default è A0, quindi nessuna rotazione. In alternativa, il comando TA*n* ruota un disegno di un numero specificato di gradi, dove *n* è un intero che va da -360 a 360. La rotazione regolata è in senso orario e quella positiva in senso antiorario.
- Il comando X disegna una sottostringa di comandi. La funzione VARPTR$ può essere richiesta per una gestione valida dei comandi della sottostringa in un programma compilato.

I comandi nella stringa DRAW possono essere scritti con lettere maiuscole o minuscole. Si possono inserire tutti i comandi che si desiderano.

Alcuni esempi di DRAW

Il programma *DateStoriche*, presentato nel Capitolo 32, visualizza sullo schermo una faccia sorridente o triste, in base alla risposta corretta o errata del giocatore. (Questi disegni appaiono solo nel modo "muto" del programma, in cui non viene emesso nessun suono.) La faccia è costituita da un cerchio grande e da due piccoli cerchi per gli occhi. Inoltre il sorriso e la bocca triste sono curve rappresentate da stringhe del comando DRAW. (Consultare il Capitolo 32, Fig. 1 e 2.). I singoli comandi compaiono in una

sequenza di istruzioni DATA alla fine della sezione principale del programma:

```
DATA 4
DATA F1 D1 F1 D1 F1 D1 F1 D1 F1 D1 F5
DATA F1 R1 F1 R1 F1 R1 F1 R1 F1 R1 R10
DATA E1 R1 E1 R1 E1 R1 E1 R1 E1 R1 E5
DATA E1 U1 E1 U1 E1 U1 E1 U1 E1 U1
```

La procedura *Faccia* legge questi comandi e li memorizza in una variabile stringa chiamata *disLin$*. La procedura riceve un argomento intero, *att%*, che indica se la faccia deve essere sorridente (*att%* è 0) o triste (*att%* è 2). Le seguenti istruzioni DRAW determinano la posizione corretta e l'angolo di rotazione del disegno per un sorriso o una faccia triste:

```
IF att% = tris% THEN DRAW "BM+50,+15"
DRAW "A=" + VARPTR$(att%) + " C4"
```

Poi questa istruzione disegna la faccia triste o sorridente:

```
DRAW disLin$
```

Le voci PSET e PRESET presentano un semplice programma che esegue un disegno che si esegue interattivamente sullo schermo.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione DRAW.

GET E PUT

L'istruzione GET prende un'immagine grafica dallo schermo corrente e la memorizza in un vettore che può essere usato in una successiva istruzione PUT. PUT visualizza l'immagine catturata in una specifica posizione sul video.

Per catturare le immagini:

```
GET (x1, y1)-(x2, y2), vett(iniz) ' (iniz) è opzionale.
GET (x1, y1)-STEP(x2, y2), vett   ' STEP indica un indirizzo relativo.
```

Per visualizzare le immagini:

```
PUT (x, y), vett(iniz), modo      ' (iniz) e modo sono opzionali.
PUT STEP(x, y), vett, modo        ' STEP indica un indirizzo relativo.
```

Insieme, le istruzioni GET e PUT eseguono molto velocemente le operazioni grafiche del tipo necessario per rappresentare sullo schermo immagini animate. L'istruzione GET è il primo passo. GET cattura l'immagine visualizzata sul video in un'area rettangolare e memorizza l'immagine in un vettore numerico. L'istruzione PUT può poi usare questo vettore per visualizzare la figura in un punto dello schermo, o per "cancellare" un'immagine uguale visualizzata precedentemente.

La sintassi dell'istruzione GET

L'istruzione GET specifica l'area rettangolare bersaglio sullo schermo come *(x1, y1)-(x2, y2)*, dove le due coppie di coordinate rappresentano angoli opposti dell'area. (Si può usare la parola chiave STEP prima di entrambe le coordinate per specificare un indirizzo relativo dall'ultimo punto cui si è fatto riferimento.) Dopo queste coordinate, GET identifica il vettore in cui l'immagine verrà memorizzata. Il vettore può appartenere ad un qualsiasi tipo numerico e deve essere definito in un'istruzione DIM prima dell'istruzione GET stessa. Le dimensioni del vettore necessarie per memorizzare l'immagine dipendono dal corrente SCREEN, dalle dimensioni dell'area rettangolare e dal tipo numerico di dati del vettore. (L'algoritmo specifico per calcolare le dimensioni appropriate del vettore viene trattato nel paragrafo "Un esempio di GET e PUT" qui di seguito.) È possibile memorizzare più di una immagine in un dato vettore. In questo caso, il punto d'inizio nel vettore a partire dal quale memorizzare una particolare figura viene specificato come *vett(iniz)*, dove *iniz* è un indice nel vettore.

La sintassi dell'istruzione PUT

L'istruzione PUT specifica un solo indirizzo dello schermo *(x, y)*, che rappresenta l'angolo in alto a sinistra dell'area in cui l'immagine verrà visualizzata. Dopo questo indirizzo, PUT identifica il vettore che contiene l'immagine grafica, catturata da una precedente istruzione GET. Infine, l'argomento opzionale *mod* nell'istruzione GET indica come la figura verrà visualizzata sullo schermo. Esistono cinque opzioni di *mod*, ognuno rappresentata da una parola chiave QuickBASIC:

XOR(il default) incorpora l'immagine nello sfondo esistente. Una seconda operazione XOR cancella la figura, lasciando l'area del video come era prima della prima operazione XOR. PSET visualizza l'immagine esattamente come memorizzata, senza considerare le figure esistenti nell'area bersaglio dello schermo.

PRESET visualizza un'immagine negativa.

OR sovrappone la figura sul contenuto attuale dell'area sullo schermo.

AND fonde l'immagine nel contenuto corrente dell'area.

Un esempio di GET e PUT

Il seguente programma è una semplice dimostrazione delle istruzioni GET e PUT. Utilizza due figure prese dal programma *DateStoriche* (presentato nel Capitolo 32), specificatamente una faccia sorridente ed una triste. Le immagini sono disegnate sullo schermo e catturate una dopo l'altra da un paio di istruzioni GET. Poi il programma usa delle istruzioni PUT all'interno di cicli nidificati per creare un modello a scacchiera di facce sullo schermo. Infine, un ciclo DO esterno cancella e rivisualizza il modello ripetutamente finché non si preme la barra spaziatrice:

```
' GETPUT.BAS
' Una dimostrazione delle istruzioni GET e PUT.

DECLARE FUNCTION DimGet% (x1%, y1%, x2%, y2%, modo%)
DECLARE SUB Facc (att%)

CONST sorr = 0
```

```
CONST tris = 2
CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST modSch% = 9

SCREEN modSch%
COLOR 1, 15

' Determina le dimensioni del vettore necessarie per l'istruzione GET.

dim% = DimGet%(324, 164, 406, 246, modSch%) / 2
DIM immFac(dim%)

' DRAW e poi GET la faccia sorridente.

Faccia sorridente
GET (324, 164)-(406, 246), immFac
CLS

' DRAW e poi GET la faccia triste.

Faccia triste
GET (324, 164)-(406, 246), immFac(dim% / 2 + 1)
CLS

' PUT crea un'organizzazione a scacchiera delle facce
' sullo schermo. Poi usa il modo XOR per cancellarle e
' rivisualizzarle ripetutamente finché l'utente non
' preme la barra spaziatrice.

prvol% = TRUE
DO
   buonUm% = FALSE
   FOR x% = 15 TO 525 STEP 85
      buonUm% = NOT buonUm%
      FOR y% = 0 TO 255 STEP 85
         IF buonUm% THEN
            PUT (x%, y%), immFac
         ELSE
            IF prVol% THEN
               PUT (x%, y%), immFac(dim% / 2 + 1), PRESET
            ELSE
               PUT (x%, y%), immFac(dim% / 2 + 1)
            END IF
         END IF
         buonUm% = NOT buonUm%
      NEXT y%
   NEXT x%
   prVol% = FALSE
LOCATE 25, 28
```

```
PRINT "Premi la barra spaziatrice per uscire.";
SLEEP
SCREEN 0

END

' Queste linee DATA contengono istruzioni DRAW per
' la faccia triste e sorridente.

DATA 4
DATA F1 D1 F1 D1 F1 D1 F1 D1 F1 D1 F5
DATA F1 R1 F1 R1 F1 R1 F1 R1 F1 R1 R10
DATA E1 R1 E1 R1 E1 R1 E1 R1 E1 R1 E5
DATA E1 U1 E1 U1 E1 U1 E1 U1 E1 U1

SUB Faccia (att%)

' Il sottoprogramma Faccia disegna sullo schermo una
' faccia sorridente o una triste.

' Leggere le istruzioni DRAW dalle linee DATA.

   RESTORE
   disLin$ = ""
   READ linInf%
   FOR i% = 1 TO linInf%
      READ d$
      disLin$ = disLin$ + d$ + " "
   NEXT i%

' Disegna la faccia.

   PSET (339, 204)
   IF att% = tris% THEN DRAW "BM+50,+15"
   DRAW "A=" + VARPTR$(att%) + " C4"
   DRAW dislIn$

   CIRCLE (355, 190), 3, 4
   CIRCLE (375, 190), 3, 4
   CIRCLE (365, 205), 40, 4
   PAINT (365, 190), 15, 4

END SUB

FUNCTION DimGet% (x1%, y1%, x2%, y2%, modo%)

' La funzione GetDim% calcola il numero di byte
' necessario per memorizzare un'immagine in un
' vettore GET.
```

```
' I valori di default:

   bitPerPix% = 1
   sup% = 1

' Gestione di SCREEN che non corrispondono al
' default:

   SELECT CASE modo%
      CASE 1
         bitPerPix% = 2
      CASE 7, 8, 9, 12
         sup% = 4
      CASE 10
         sup% = 2
      CASE 13
         bitPerPix% = 8
   END SELECT

' Calcola la larghezza e la lunghezza dell'area grafica.

   deltaX% = ABS(x1% - x2%) + 1
   deltaY% = ABS(y1% - y2%) + 1

' Esegue il calcolo principale.

   fatBit% = (deltaX% * bitPerPix% + 7) / 8
   DimGet% = 4 + INT(fatBit% * sup%*deltaY%)

END FUNCTION
```

Il programma contiene una procedura chiamata *Fac*, che semplicemente disegna sullo schermo le facce originali così che le immagini possano essere catturate dalle istruzioni GET:

```
Faccia sorridente
GET (324, 164)-(406, 246), immFac
CLS

Faccia triste
GET (324, 164)-(406, 246), immFac(dim% / 2 + 1)
```

Si noti che le immagini sono memorizzate in due metà dello stesso vettore numerico in precisione semplice, chiamato *immFac*. Una chiamata della funzione *DimGet%* determina le dimensioni appropriate per questo vettore.

DimGet% prende cinque argomenti interi: le coordinate *x* e *y* per due angoli opposti dell'area bersaglio dello schermo ed il numero di byte necessari per memorizzare l'immagine:

```
deltaX% = ABS(x1% - x2%) + 1
deltaY% = ABS(y1% - y2%) + 1

fatBit% = (deltaX% * bitPerPix% + 7) / 8
DimGet% = 4 + INT(fatBit% * sup%*deltaY%)
```

Le variabili *deltaX%* e *deltaY%* rappresentano la larghezza e la lunghezza dell'area rettangolare. I valori rappresentati da *bitPerPix%* e *sup%* sono i fattori dipendenti da SCREEN che *GetDim%* calcola partendo dal valore dell'argomento inviato come modo schermo. Ecco come il programma chiama la funzione *GetDim%* e poi definisce il vettore *immFac*:

```
dim% = GetDim%(324, 164, 406, 246, modSch%) / 2
DIM immFac(dim%)
```

GetDim% calcola il numero totale di byte richiesti per memorizzare un'immagine. Quattro byte di memoria sono necessari per ogni elemento di un vettore a precisione semplice, ma *immFac* deve tenere due immagini complete di *dim%* byte ciascuno. Il programma perciò divide il valore *GetDim%* per 2 per determinare le giuste dimensioni del vettore in questo contesto. Si tenga presente che si può usare un vettore di qualsiasi tipo numerico per catturare un'immagine. Le dimensioni degli elementi del vettore variano secondo il tipo di dati: due byte per elemento per un vettore intero; quattro byte per gli elementi del vettore interi lunghi e a precisione semplice ed otto byte per un elemento del vettore a doppia precisione. Il programma sceglie tra tre diverse istruzioni PUT per visualizzare le facce sullo schermo. Questa istruzione mostra la faccia sorridente sul video nel suo modo schermo originale:

```
PUT (x%, y%), immFac
```

Al contrario, la faccia triste inizia in mezzo al vettore *immFac* ed è visualizzata come un'immagine negativa:

```
PUT (x%, y%), immFac(dim% / 2 + 1), PRESET
```

In seguito, le stesse immagini vengono cancellate e rivisualizzate ripetutamente all'interno di un ciclo DO. Questo processo viene trasferito in un modo XOR per default dell'istruzione PUT:

```
PUT (x%, y%), immFac(dim% / 2 + 1)
```

Compatibilità

Sia Turbo Basic sia BASICA hanno le istruzioni GET e PUT.

LINE

L'istruzione LINE disegna una linea orizzontale, verticale o diagonale sullo schermo. In alternativa, disegna il contorno di un rettangolo e facoltativamente lo riempie con il colore.

```
LINE (x1, y1)-(x2, y2), col         ' Disegna una linea sullo schermo.
                                    ' col è opzionale.

LINE (x1, y1)-STEP (x2, y2)         ' Usa indirizzi relativi.

LINE (x1, y1)-(x2, y2), col, BF     ' Disegna un rettangolo (B) e
                                    ' facoltativamente lo riempie.

GET (x1, y1)-(x2, y2), col,, trat   ' Usa una maschera per disegnare
                                    ' una linea tratteggiata.
```

Gli indirizzi *(x1,y1)-(x2,y2)* nell'istruzione LINE esprimono il punto d'inizio e di fine della linea risultante. Li si può esprimere come indirizzi assoluti all'interno dell'area grafica corrente oppure come indirizzi relativi, usando la parola chiave STEP. (Un indirizzo relativo è un numero specificato di pixel orizzontali e verticali dall'indirizzo dello schermo a cui si è fatto riferimento precedentemente.) L'argomento opzionale *col* rappresenta l'attributo del colore della linea; se lo si omette, la linea verrà disegnata nel colore di primo piano attuale. L'opzione B fa sì che LINE disegni un rettangolo invece di una linea. In questo caso, i due indirizzi diventano gli angoli opposti del rettangolo risultante. L'opzione BF riempie il rettangolo con il colore indicato. L'opzione *trat* è un intero che QuickBASIC usa nella

forma binaria da un tratteggio per la linea finale. Ogni bit di valore 1 nel numero binario corrisponde ad un punto sulla linea ed ogni posizione 0 è uno spazio vuoto. Questa opzione vale per le linee e i contorni del rettangolo, ma non per i rettangoli pieni. In altre parole, se si utilizza l'opzione BF, l'argomento *trat* viene ignorato. (Si può usare l'istruzione PAINT per riempire un'area delimitata con un tratteggio definito da cifre binarie.)

Alcuni esempi di LINE

Il programma *DateStoriche*, listato nel Capitolo 32, usa il comando LINE per disegnare la seconda lancetta di un orologio e le finestre di testo visualizzate sullo schermo. La posizione della seconda lancetta deve essere ricalcolata ogni volta che la lancetta si muove lungo la circonferenza dell'orologio. Ecco il frammento che esegue i calcoli e disegna la linea:

```
ang% = ang% + incrAng%
xCirc = lunghlanc% * COS(Rad(ang%))
yCirc = lunghlanc% * SIN(Rad(ang%))
LINE (0, 0)-(xCirc, yCirc), 4
```

Il programma visualizza anche una finestra nella parte superiore dello schermo per racchiudere i messaggi rivolti all'utente. Questo passaggio disegna la finestra e visualizza la richiesta all'interno di essa:

```
LINE (47, 63)-(582, 92), 4, B
PAINT (48, 64), 15, 4
LOCATE 6, 8
PRINT infData(numRich%).even
```

In questo caso, il programma usa l'istruzione PAINT invece dell'opzione BF per riempire con il colore la finestra, perché il colore scelto è diverso da quello del contorno del rettangolo. Per usare l'opzione *trat* dell'istruzione LINE è utile avere uno strumento che converte una stringa di cifre binarie in un intero. La funzione che segue ha questo ruolo; prende una stringa di caratteri "0" e "1" come proprio argomento e ritorna un intero in base 10:

```
FUNCTION BinInDec% (inBin$)
   bit% = LEN(inBin$)
   IF bit% > 15 THEN
```

```
      inBin$ = RIGHT$(inBin$, 15)
      bit% = 15
   END IF

   valCor% = 0
   FOR i% = bit% TO 1 STEP -1
      cifCor% = VAL(MID$(inBin$, i%, 1))
      IF cifCor% > 1 THEN cifCor% = 1
      valCor% = valCor% + cifCor% * 2 ^ (bit% - i%)
   NEXT i%

   BinInDec% = valCor%

END FUNCTION
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione LINE.

PSET E PRESET

Le istruzioni PSET e PRESET visualizzano singoli punti su uno schermo grafico.

```
PSET (x, y), col           ' Visualizza il punto (x, y). Se col è omesso,
                           ' usa il colore di primo piano.

PSET STEP (x, y), col      ' Visulizza il punto in un indirizzo relativo.

PRESET (x, y), col         ' Se col viene omesso, usa il colore di sfondo.

PRESET STEP (x, y), col    ' Visualizza il punto in un indirizzo relativo.
```

La posizione del punto viene definita o come indirizzo assoluto *(x, y)* o come indirizzo relativo *STEP (x, y)*. (Un indirizzo relativo è un movimento in avanti o indietro partendo dall'ultimo indirizzo a cui si era fatto riferimento sullo schermo.) L'argomento *col* indica l'attributo del colore con cui il punto verrà visualizzato. Se si omette *col*, PSET visualizza il punto nel colore di primo piano corrente. Al contrario, PRESET usa il colore attuale di sfondo per "cancellare" il punto. Se si include l'argomento opzionale *col*, PSET e PRESET sono identici.

Un esempio di PSET

Questo programma, chiamato *Disegna,* crea un file del disco dei comandi DRAW di stringa, convertito dalle operazioni interattive di disegno sullo schermo:

```
' DISEGNA.BAS
' Questo programma crea un file dei comandi DRAW
' stringa.

CLS
PRINT "Creare un file dei comandi DRAW stringa"
PRINT
INPUT "Immettere il nome del file che si vuole usare: ", nomeFile$
INPUT "Immettere il modo schermo in cui si vuole lavorare: ", modSel%
PRINT
PRINT "Il disegno inizia al centro del video."
PRINT "Per disegnare usare i tasti con le frecce del
         tastierino numerico della tastiera."
INPUT "Premere <Invio> per iniziare, poi premere <Q> per uscire.", go$

OPEN nomeFile$ FOR OUTPUT AS #1

SCREEN modSel%

i% = 0
DO WHILE inDir$ <> "Q"
   inDir$ = UCASE$(INKEY$)
   IF LEN(inDir$) = 2 THEN
      i% = i% + 1
      tastDir% = ASC(RIGHT$(inD ir$, 1))
      SELECT CASE tastDir%
         CASE 71
            PSET STEP(-1, -1)
            PRINT #1, "H ";
         CASE 72
            PSET STEP(0, -1)
            PRINT #1, "U ";
         CASE 73
            PSET STEP(1, -1)
            PRINT #1, "E ";
         CASE 75
            PSET STEP(-1, 0)
            PRINT #1, "L ";
         CASE 77
            PSET STEP(1, 0)
            PRINT #1, "R ";
```

```
        CASE 79
           PSET STEP(-1, 1)
           PRINT #1, "G ";
        CASE 80
           PSET STEP(0, 1)
           PRINT #1, "D ";
        CASE 81
           PSET STEP(1, 1)
           PRINT #1, "F ";
     END SELECT
  END IF
LOOP

CLOSE #1
SCREEN 0
```

Il programma inizia chiedendo un nome del file e un modo schermo. Dopo aver aperto il file per l'output, il programma passa al modo schermo indicato ed aspetta i comandi della tastiera. Si usano gli otto tasti con le frecce del tastierino numerico per creare una figura sullo schermo. Ogni volta che si preme uno di questi tasti, il programma disegna un nuovo punto sul video ed invia l'equivalente comando stringa DRAW nel file del disco. Ogni punto viene disegnato in un indirizzo relativo sul video. Ad esempio, ecco l'operazione che avviene quando si preme Home:

```
PSET STEP(-1, -1)
PRINT #1, "H ";
```

Poi il comando H fa sì che con l'istruzione DRAW ci si muova diagonalmente, in alto e verso sinistra. Si può usare il file di testo creato da *Disegna* in diversi modi. Una tecnica è quella di incorporare il file direttamente in un listato del programma QuickBASIC, usando il comando Merge nel menù di File. Un'altra tecnica è quella di scrivere una routine che semplicemente legge il file e ne memorizza il contenuto in una variabile stringa. In seguito, un'istruzione DRAW riprodurrà il disegno originale dello schermo.

Compatibilità

Sia Turbo Basic sia BASICA hanno i comandi PSET e PRESET.

VARPTR$

La funzione VARPTR$ ritorna l'indirizzo di una variabile, in un formato stringa che è utilizzabile nelle istruzioni DRAW e PLAY.

```
VARPTR$(var)
```

VARPTR$ incorpora una sottostringa nella stringa di comandi di un'istruzione DRAW o PLAY. L'argomento di questa funzione è il nome di una variabile numerica o di una stringa di comandi.

Un esempio di VARPTR$

Il programma *DateStoriche*, listato nel Capitolo 32, contiene un esempio di VARPTR$ in un sottoprogramma chiamato *Faccia*. Questa routine consente di disegnare sullo schermo una faccia triste o sorridente, in base alla risposta corretta o errata che il giocatore dà alla domanda sottoposta. Una variabile intera chiamata *att%* contiene un valore pari a 0 o 2, che rappresenta un angolo di rotazione di 0 o 180 gradi, per disegnare la faccia sorridente o triste. Dato questo valore, la seguente istruzione DRAW disegna l'espressione sorridente o triste sulla faccia:

```
DRAW "A=" + VARPTR$(att%) + " C4"
```

Per ulteriori informazioni sulle stringhe di comando ritornare alla voce DRAW.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione VARPTR$, ma i contesti in cui la funzione si può utilizzare variano da un ambiente di programmazione all'altro.

Parte V

Funzioni numeriche e stringa

Le funzioni *incorporate* di QuickBASIC sono strumenti predefiniti che ritornano risultati di calcoli o eseguono utili operazioni su argomenti specificati. La Parte V descrive le tante funzioni incorporate che si possono usare per lavorare con le stringhe e valori di tipo numerico. (Le funzioni progettate per altri scopi speciali vengono trattate altrove in questo libro; ad esempio nella Parte III sono descritte le funzioni di input e output e nella Parte IV le funzioni relative alla grafica. Inoltre, le funzioni per la gestione di data e ora vengono esposte nella Parte VII.)

Le applicazioni che lavorano estensivamente con i caratteri e il testo richiedono tecniche efficienti per eseguire complesse manipolazioni sulle stringhe. Per questi programmi, il QuickBASIC fornisce una serie di strumenti che eseguono molte operazioni su stringa, tra cui:

- Analisi del contenuto di una stringa (la funzione INSTR)
- Creazione di stringhe di caratteri ripetuti (le funzioni SPACE$ e STRING$)
- Variazione delle lettere di una stringa da maiuscolo a minuscolo e viceversa (le funzioni UCASE$ e LCASE$)
- Eliminazione degli spazi iniziali o terminali da una stringa (le funzioni LTRIM$ e RTRIM$)
- Determinazione della lunghezza di una stringa (la funzione LEN)

- Modifica del contenuto di una stringa (l'istruzione MID$)
- Analisi degli elementi che compongono una stringa (le funzioni LEFT$, RIGHT$ e MID$)
- Manipolazione dei codici ASCII (le funzioni ASC e CHR$)

I primi due capitoli della Parte V descrivono questi strumenti e analizzano i modi specifici in cui essi possono essere utilizzati. Il Capitolo 18 presenta le funzioni (ed un'istruzione) connesse all'elaborazione delle sottostringhe e il Capitolo 19 descrive molti altri tipi di funzioni stringa. Inoltre, il QuickBASIC fornisce due funzioni relative alle stringhe, STR$ e VAL, che convertono i valori di tipo numerico in stringhe e viceversa. Dettagliate informazioni su queste due funzioni si trovano nel Capitolo 21, "Conversioni di Tipi di Dati".

QuickBASIC fornisce anche molte importanti funzioni numeriche e matematiche. Sono comprese le funzioni trigonometriche, esponenziali, logaritmiche e di segno oltre a strumenti che generano numeri casuali e le conversioni di base numerica; tutte descritte nel Capitolo 20. Infine il Capitolo 21 analizza un gruppo interessante di funzioni che ritornano valori convertiti da tipi di dati diversi.

Capitolo 18

Funzioni di gestione delle stringhe

Una sottostringa è una sequenza di caratteri contenuti all'interno di un'altra stringa. QuickBASIC offre tre funzioni, LEFT$, RIGHT$ e MID$, che ritornano sottostringhe copiate da posizioni specifiche nelle stringhe esistenti. Inoltre, la funzione INSTR determina la posizione di una sottostringa data all'interno di una stringa. Infine, l'istruzione MID$ (da non confondere con la funzione MID$) rimpiazza una sequenza in una stringa con una sottostringa. Questi strumenti, ognuno dei quali è disponibile nella maggior parte delle versioni di BASIC, sono particolarmente importanti nei programmi che devono esaminare o *analizzare* il contenuto dei valori stringa d'input. In questo capitolo vi sono molti esempi di questo tipo di applicazione.

INSTR

La funzione INSTR ritorna un intero che rappresenta la posizione di una sottostringa all'interno di una stringa. Se nella stringa esaminata non vi è una sottostringa, INSTR ritorna un valore pari a 0.

```
INSTR(posiz, str, sottostr)  ' posiz è opzionale.
```

L'argomento opzionale *posiz* è un intero che specifica la posizione del carattere in *str* dove inizierà la ricerca di *sottostr*. Se si omette l'argomento *posiz*, la ricerca di *sottostr*, parte dall'inizio di *str*. Se si trova *sottostr*, INSTR ritorna la posizione del suo primo carattere in *str*. In altre tre situazioni

INSTR ritorna il valore 0: se l'argomento *sottostr* non viene trovato, se l'argomento *str* è una stringa vuota o se *posiz* è maggiore della lunghezza di *str*. D'altra parte, se *sottostr* è una stringa vuota, INSTR ritorna il valore 1 o il valore dell'argomento *posiz* se è presente. I tre argomenti di INSTR possono presentarsi come valori literal, variabili o espressioni.

Alcuni esempi di INSTR

INSTR è utile per analizzare il contenuto dei valori stringa d'input. Ad esempio, il programma *Mese* (presentato nel Capitolo 29) utilizza la seguente istruzione per esaminare la linea di comando che l'utente ha immesso dal prompt del DOS:

```
posVirg% = INSTR(linCom$, ",")
```

Se l'argomento del prompt del DOS dell'utente contiene una virgola, la posizione della virgola viene memorizzata nella variabile intera *posVirg%*. Se non vi è la virgola, a *posVirg%* viene assegnato il valore 0. Quando un programma prende un solo carattere dall'utente a tastiera, un carattere che rappresenta una selezione del menù, ad esempio, la funzione INSTR può essere usata per commutare il carattere d'input in un intero. Ad esempio, il programma *Compleanno* (presentato nel Capitolo 28) dà all'utente una selezione di ordini in cui si può organizzare una tabella d'output:

```
Gratifiche di compleanno dei dipendenti per il 1990.
Organizzare la tabella in base a:

N)omi.
Q)ualifica.
A)nzianità (ordine ascendente).
D)iscendente (Anzianità).
E)tà (dei dipendenti).
C)ompleanni.

   --> _
```

L'utente esegue una selezione premendo un solo tasto contrassegnato dalla lettera: N, Q, A, D, E o C. Questo ciclo controlla il processo d'input:

```
DO
   scelta$ = UCASE$(INKEY$)
   scelta% = INSTR("NQADEC", scelta$)
LOOP WHILE scelta$ = "" OR scelta% < 1
```

Il programma memorizza la versione maiuscola del carattere immesso dall'utente nella variabile stringa *scelta$*. La funzione INSTR trova poi la posizione di *scelta$* in una stringa delle opzioni del menù, "NQADEC". (Il ciclo DO continua ad aspettare l'input fin che l'utente non effettua una scelta valida del menù.) Il programma successivamente usa il valore intero di *scelta%* per selezionare un ordine per la tabella d'output. Un esempio finale di INSTR è preso anche dal programma *Compleanno*. Questo frammento analizza la risposta "S" o "N" che l'utente dà premendo solo i tasti S o N:

```
PRINT "Visualizzo un'altra tabella? <S> o <N> ";

DO
   risp$ = UCASE$(INKEY$)
LOOP UNTIL risp <> "" AND INSTR("SN", risp$) <> 0

Fine% = (risp$ = "N")
```

In questo caso, INSTR ritorna 0 fin che l'utente non preme il tasto S o N. Si noti che la condizione del ciclo deve includere anche un test che determina se *risp$* è una stringa vuota. INSTR ritorna il valore errato 1 se l'utente non ha premuto nessun tasto e *risp$* perciò è ancora vuoto.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione INSTR.

LEFT$

La funzione LEFT$ ritorna una sottostringa copiata dall'inizio di una stringa.

```
LEFT$(str, lungh)
```

L'argomento *lungh* è un intero che indica il numero di caratteri che LEFT$ ritornerà dall'inizio dell'argomento *str*. Se *lungh* è maggiore della lunghezza di *str*, LEFT$ ritorna la stringa intera. Entrambi gli argomenti possono comparire come valori literal, variabili o espressioni.

Un esempio di LEFT$

Questo esempio è preso dal programma *Mese* (presentato nel Capitolo 29). Il compito di questa istruzione è isolare la prima parte dell'argomento della linea di comando che l'utente dà dal prompt del DOS:

```
mese% = VAL(LEFT$(linCom$, posVirg%))
```

La funzione LEFT$ ritorna i caratteri iniziali dell'argomento della linea di comando (fino alla posizione di *posVirg%* di una virgola) e la funzione VAL converte queste cifre in un valore intero.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione LEFT$.

LA FUNZIONE MID$

La funzione MID$ ritorna una sottostringa di caratteri copiata da una posizione specificata in una stringa.

```
MID$(str, posiz, lungh)   ' lungh è opzionale.
```

L'argomento *posiz* è un intero che specifica la posizione di un carattere in *str*. L'argomento *lungh* indica il numero di caratteri che MID$ copierà, iniziando da *posiz*. Se si omette l'argomento *lungh*, MID$ ritorna una stringa composta da tutti i caratteri da *posiz* alla fine di *str*. I tre argomenti possono comparire come valori literal, variabili o espressioni.

Un esempio della funzione MID$

Questo esempio è preso dal programma *Agenda* (presentato nel Capitolo 30). L'istruzione prende informazioni dall'argomento della linea di comando che l'utente fornisce al programma dal prompt del DOS:

```
anno% = VAL(MID$(inData$, posSlash2% + 1))
```

Il programma si aspetta che l'utente fornisca una stringa di dati, *inData$*, che è delimitata da slash. La funzione MID$ in questa istruzione copia tutti i caratteri collocati dopo il secondo slash e la funzione VAL li converte in un intero. Si noti che il terzo argomento opzionale viene omesso nell'esempio MID$.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione MID$.

L'ISTRUZIONE MID$

L'istruzione MID$ scrive una stringa di riposizionamento su una lunghezza di caratteri in una variabile stringa.

```
MID$(var, posiz, lungh) = str  ' lungh è opzionale.
```

Il primo argomento dell'istruzione MID$ è una variabile stringa esistente il cui valore attuale ha una lunghezza di uno o più caratteri. Il secondo argomento, *posiz* è un intero che specifica il punto di inizio da cui partirà il riposizionamento nella variabile stringa. (Il valore di *posiz* deve essere minore o uguale alla lunghezza della stringa, altrimenti il programma darà un errore run-time, "chiamata di funzione impossibile".) Il terzo argomento, *lungh* indica il numero di caratteri che sarà riposizionato in una variabile stringa. La stringa di riposizionamento appare a destra del segno di uguale. Se si omette l'argomento *lungh*, la lunghezza di *str* determina il numero di caratteri che verrà riposizionato, ad eccezione di MID$ che non aumenta la lunghezza corrente di *var*. Gli argomenti *posiz, lungh* e *str* possono comparire come valori literal, variabili o espressioni.

Un esempio dell'istruzione MID$

I paragrafi "Compatibilità" delle voci UCASE$ e LCASE$ presentano delle routine per eseguire le conversioni alfabetiche in minuscolo o maiuscolo in un BASIC che non ha le funzioni UCASE$ e LCASE$ incorporate. Queste due routine illustrano l'utilizzo dell'istruzione MID$. Ad esempio, questo frammento converte una stringa memorizzata nella variabile *st$* per tutte le lettere maiuscole:

```
diff% = ASC("a") - ASC("A")

FOR car% = 1 TO LEN(st$)
   car$ = MID$(st$, car%, 1))
   IF car$ >= "a" AND car$ <= "z" THEN
      car$ = CHR$(ASC(car$) - diff%)
      MID$(st$, car%, 1) = car$
   END IF
NEXT car%
```

La conversione avviene all'interno di un ciclo FOR che esamina ogni carattere di una valore stringa memorizzato nella variabile *st$*. Per ogni carattere che è una lettera minuscola (compresa tra "a" e "z"), il programma usa le funzioni CHR$ e ASC per trovare la lettera maiuscola equivalente. Poi la seguente istruzione MID$ rimpiazza la lettera minuscola con la maiuscola:

```
MID$(st$, car%, 1) = car$
```

Questa istruzione rimpiazza un carattere in *st$* con l'appropriata lettera maiuscola.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione MID$.

RIGHT$

La funzione RIGHT$ ritorna una sottostringa copiata dalla fine di una stringa.

```
RIGHT$(str, lungh)
```

L'argomento *lungh* è un intero che indica il numero di caratteri che RIGHT$ ritornerà dalla fine dell'argomento *str*. Se *lungh* è maggiore della lunghezza di *str*, RIGHT$ ritorna la stringa intera. Entrambi gli argomenti possono comparire come valori literal, variabili o espressioni.

Un esempio di RIGHT$

Questo esempio è preso dal programma *Mese* (presentato nel Capitolo 30). Il compito di questa istruzione di decisione è quello di esaminare la parte finale dell'argomento della linea di comando che l'utente fornisce dal prompt del DOS:

```
IF RIGHT$(linCom$, 2) = "/P" THEN
   qualeEmis% = qualeEmis% + infStampa%
END IF
```

La funzione RIGHT$ ritorna i due caratteri finali dell'argomento della linea di comando e l'istruzione IF guarda se i caratteri sono "/P". Il programma legge questi caratteri come un'istruzione per inviare l'output alla stampante piuttosto che allo schermo.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione RIGHT$.

Capitolo 19

Altre funzioni stringa

QuickBASIC fornisce altre funzioni stringa in molte diverse categorie:

- Le funzioni ASC e CHR$ danno l'accesso al codice ASCII.
- Le funzioni LCASE$ e UCASE$ ritornano copie minuscole o maiuscole dei loro argomenti stringa.
- La funzione LEN determina la lunghezza di una stringa.
- Le funzioni LTRIM$ e RTRIM$ eliminano gli spazi iniziali o finali dalla copia di una stringa.
- Le funzioni SPACE$ e STRING$ creano delle stringhe costituite da caratteri ripetuti.

Questo capitolo descrive tutte queste funzioni e suggerisce molte loro applicazioni. Alcune di queste funzioni sono relativamente nuove per il BASIC Microsoft; esse non erano disponibili in BASICA e nemmeno nelle prime versioni dello stesso QuickBASIC. Inoltre per dare le illustrazioni di queste funzioni, questo capitolo suggerisce alcune tecniche per eseguire operazioni stringa equivalenti nei BASIC che non hanno questi utili strumenti.

ASC

La funzione ASC ritorna il numero di codice ASCII di un carattere, un intero da 0 a 255.

```
ASC(str)
```

L'argomento stringa di ASC può essere espresso come un valore literal, una variabile o un'espressione. ASC dà il codice ASCII del primo carattere della stringa. Solitamente l'argomento di ASC è un solo carattere, ma la stringa può avere qualsiasi lunghezza maggiore di 0. Se l'argomento è una stringa vuota, la chiamata ASC dà un messaggio d'errore run-time, "Chiamata di funzione errata". Vi è una tabella ASCII nell'Appendice A.

Un esempio di ASC

I paragrafi "Compatibilità" delle voci UCASE$ e LCASE$ presentano delle routine per eseguire conversioni alfabetiche in maiuscolo o minuscolo in un BASIC che non ha le funzioni incorporate LCASE$ e UCASE$. Queste routine illustrano l'uso della funzione ASC. Ad esempio, il seguente frammento converte in tutte lettere maiuscole una stringa memorizzata nella variabile *st*:

```
diff% = ASC("a") - ASC("A")

FOR car% = 1 TO LEN(st$)
   car$ = MID$(st$, car%, 1)
   IF car$ >= "a" AND car$ <= "z" THEN
      car$ = CHR$(ASC(car$) - diff%)
      MID$(st$, car%, 1) = car$
   END IF
NEXT car%
```

I numeri di codice ASCII per le lettere maiuscole (da 65 a 90) sono minori di quelli per le lettere minuscole (da 97 a 122). Sottraendo il codice per "A" dal codice per "a" dà perciò un intero *diff%* che può essere usato per convertire le lettere minuscole in maiuscolo:

```
diff% = ASC("a") - ASC("A")
```

Il ciclo FOR in questa routine esamina ogni carattere nella lunghezza della stringa *st$*. Se si trova un dato carattere nell'intervallo delle lettere minuscole, la seguente istruzione esegue la conversione:

```
car$ = CHR$(ASC(car$) - diff%)
```

Il valore di *diff%* è sottratto dal codice ASCII del carattere minuscolo *car$*. Poi la funzione CHR$ dà la lettera maiuscola corrispondente. Infine, un'istruzione MID$ inserisce il carattere convertito nella precedente posizione nella stringa *st$*:

```
MID$(st$, car%, 1) = car$
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione ASC.

CHR$

La funzione CHR$ ritorna il carattere che corrisponde ad un codice ASCII che va da 0 a 255.

```
CHR$(num)
```

L'argomento intero di CHR$ può essere espresso come un valore literal, una variabile o un'espressione. Il risultato della funzione è una stringa che contiene un solo carattere ASCII. Se l'argomento non è compreso nell'intervallo degli interi del codice ASCII, da 0 a 255, la chiamata a CHR$ dà un messaggio di errore run-time, "Chiamata di funzione errata". Nell'Appendice A vi è una tabella ASCII. Se si analizza questa tabella si trova un vasto gruppo di caratteri non rappresentati sulla tastiera. Ad esempio la seconda metà del codice (da 128 a 255) contiene simboli correnti, lettere greche, speciali caratteri usati in matematica ed un gruppo di caratteri della "grafica del testo" (codici da 176 a 223) che creano forme geometriche sullo schermo. La funzione CHR$ offre un utile accesso a tutti questi caratteri.

Alcuni esempi di CHR$

Il programma *GestioneDate* (presentato nel Capitolo 31) usa CHR$ per visualizzare sullo schermo molti caratteri speciali, nelle istruzioni che spiegano le funzioni tastiera del programma. Specificatamente, il programma visualizza i quattro tasti "freccia", che hanno come numeri ASCII da 24 a 27:

```
CHR$(24)    ' freccia verso l'alto.
CHR$(25)    ' freccia verso il basso.
CHR$(26)    ' freccia verso destra.
CHR$(27)    ' freccia verso sinistra.
```

Il programma stabilisce quattro costanti simboliche per rappresentare questi codici ASCII:

```
CONST sin% = 27
CONST des% = 26
CONST su% = 24
CONST giù% = 25
```

Poi, nella sequenza di istruzioni PRINT che visualizza sullo schermo le istruzioni del programma, la funzione CHR$ dà i caratteri freccia stessi:

```
PRINT CHR$(sin%); " "; CHR$(des%);
PRINT "          Selezionare un nuovo giorno."
PRINT CHR$(su%); " "; CHR$(giù%);
PRINT "          Selezionare una nuova settimana."
PRINT "Ctrl-"; CHR$(sin%); " Ctrl-"; CHR$(des%);
PRINT "          Selezionare un nuovo mese."
PRINT "Ctrl-"; CHR$(su%); " Ctrl-"; CHR$(giù%);
PRINT "          Selezionare un nuovo anno."
```

Per ulteriori informazioni sul programma *GestioneDate* e sulle istruzioni schermo create da queste istruzioni PRINT consultare il Capitolo 31. Il programma *GestioneDate* utilizza anche la funzione CHR$ nelle istruzioni KEY che definiscono la gestione degli eventi per la tastiera. La voce KEY nel Capitolo 23 descrive questo impiego dettagliatamente.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione CHR$.

LCASE$

La funzione LCASE$ ritorna l'equivalente minuscolo di una stringa.

```
LCASE$(str)
```

L'argomento *str* può presentarsi come un valore literal tra virgolette, come una variabile o come un'espressione. Le lettere maiuscole nell'argomento iniziale vengono convertite in minuscole nel valore di ritorno. Tutti gli altri caratteri vengono copiati come sono.

Un esempio di LCASE$

Un uso comune della funzione LCASE$ è semplicemente quello di introdurre la consistenza in un campo di articoli della stringa in maiuscolo e minuscolo. Ad esempio, questo esercizio visualizza una lista di stringhe dopo aver convertito ogni articolo in un formato standard maiuscolo o minuscolo:

```
READ lunghList%
PRINT "Scorte:"
PRINT "------"
FOR art% = 1 TO lunghList%
   READ art$
   PRINT UCASE$(LEFT%(art%, 1)); LCASE$(MID$(art$, 2))
NEXT art%

DATA 5
DATA CARTA
DATA Nastri
DATA DISCHI
DATA ETICHETTE
DATA Contenitori per i dischi
```

Ecco l'output del programma:

```
Scorte:
------
Carta
Nastri
Dischi
Etichette
Contenitori per i dischi
```

Come si può vedere, il programma crea delle stringhe di output in cui la prima lettera è maiuscola e tutte le altre sono minuscole.

Compatibilità

Turbo Basic ha la funzione LCASE, mentre BASICA no. In una versione di BASIC che non ha le funzioni di conversione in maiuscolo o minuscolo, si può usare questa routine per convertire tutte le lettere maiuscole della variabile stringa *st$* in minuscole:

```
diff% = ASC("a") - ASC("A")

FOR car% = 1 TO LEN(st$)
   car$ = MID$(st$, car%, 1)
   IF car$ >= "A" AND car$ <= "Z" THEN
      car$ = CHR$(ASC(car$) + diff%)
      MID$(st$, car%, 1) = car$
   END IF
NEXT car%
```

Per analizzare questa routine, vedere ASC e l'istruzione MID$.

LEN

Dato un argomento stringa, la funzione LEN ritorna la lunghezza della stringa in caratteri. LEN può anche essere usata per definire la richiesta di memoria, in byte, di un valore rappresentato da una variabile.

```
LEN(str)      ' Indica la lunghezza della stringa.

LEN(var)      ' Indica la richiesta di memoria per un valore
              ' del tipo rappresentato dalla variabile.
```

Nella prima sintassi della funzione LEN, l'argomento della stringa può presentarsi come un valore stringa literal messo tra virgolette, come una variabile o un'espressione. Data una stringa vuota, LEN ritorna un valore pari a 0. Altrimenti, LEN dà un valore intero compreso tra 1 e 32.767, la lunghezza massima di una stringa. Il risultato di LEN per una variabile stringa a lunghezza fissa è sempre la lunghezza definita della stringa, senza tener conto del fatto che un valore sia stato effettivamente assegnato alla variabile. Nella seconda sintassi di LEN, l'argomento può essere una variabile che appartiene ad un qualsiasi tipo di dati, compreso il tipo record definito dall'utente. In questo caso, LEN ritorna un intero che rappresenta il numero di byte richiesti per memorizzare un valore del tipo corrispondente.

Alcuni esempi di LEN

Il programma *Agenda* (presentato nel Capitolo 30) contiene molti esempi interessanti della funzione LEN. Per prima cosa, questo frammento usa le funzioni LEN e STRING$ per visualizzare una riga di lineette sotto la stringa della data rappresentata dalla variabile *strData$*:

```
PRINT strData$
PRINT STRING$(LEN(strData$), "-")
```

Il primo argomento della funzione STRING$ è un intero che specifica la lunghezza della stringa che la funzione creerà. In questo caso, la lunghezza è data come il risultato della funzione LEN. Insieme, queste due istruzioni producono un output come il seguente:

```
Giov., Giu. 28, 1989
--------------------
```

Il programma *Agenda* usa anche LEN per misurare la struttura record che il programma definisce per un file ad accesso casuale chiamato AGENDA.DAT. Ad esempio, questa istruzione OPEN utilizza la funzione LEN per definire la lunghezza del record del file:

```
OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = LEN(gior)
```

Qui *gior* è una variabile record che corrisponde alla struttura record del file ad accesso casuale. (Non si confonda la clausola LEN dell'istruzione OPEN

con la funzione incorporata LEN.) Una volta che il file è aperto, questa istruzione calcola il numero di record nel file:

```
contImm% = LOF(1) / LEN(gior)
```

La funzione LOF dà la lunghezza, in byte, del file. Dividendo questo numero per la lunghezza di un dato record si ha il conteggio corretto dei record. (Per informazioni più dettagliate sui file ad accesso casuale in QuickBASIC bisogna tornare al Capitolo 11.)

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione LEN, ma l'utilizzo della funzione si limita a trovare la lunghezza di una stringa.

LTRIM$

La funzione LTRIM$ ritorna una copia dell'argomento della sua stringa con gli spazi iniziali cancellati.

```
LTRIM$(str)
```

L'argomento può presentarsi come una stringa literal, una variabile o un'espressione. La stringa risultante è più corta della stringa dell'argomento del numero degli spazi iniziali contenuti nell'argomento.

Un esempio di LTRIM$

Un'istruzione nel programma *Compleanno* (presentato nel Capitolo 28) illustra l'uso di LTRIM$ per eliminare gli spazi iniziali che solitamente vi sono prima della visualizzazione di un intero positivo:

```
LTRIM$(STR$(gi%))
```

Ecco la funzione STR$ che fornisce la versione della stringa dell'intero *gi%* e LTRIM$ elimina lo spazio iniziale.

Compatibilità

Né Turbo Basic né BASICA hanno la funzione LTRIM$. In una versione di BASIC che non ha questa funzione, si può usare per sostituirla, questa routine:

```
prim% = 0
DO
   prim% = prim% + 1
   car$ = MID$(st$, prim%, 1)
LOOP WHILE car$ = " "

ltrimSt$ = MID$(st$, prim%)
```

La routine copia il valore stringa memorizzato in *st$* e poi in *ltrimSt$* ed elimina gli spazi iniziali dalla copia.

RTRIM$

La funzione RTRIM$ ritorna una copia dell'argomento della sua stringa con gli spazi finali cancellati.

```
RTRIM$(str)
```

L'argomento può presentarsi come una stringa literal, una variabile o un'espressione. La stringa risultante è più corta della stringa dell'argomento del numero degli spazi finali contenuti nell'argomento.

Un esempio di RTRIM$

Il programma *Agenda* (presentato nel Capitolo 30) usa la funzione RTRIM$ per eliminare gli spazi finali dalla stringa della linea di comando che l'utente fornisce dal prompt del DOS:

```
inData$ = RTRIM$(COMMAND$)
```

QuickBASIC elimina automaticamente gli spazi *iniziali* dalla stringa COMMAND$, ma non quelli finali.

Compatibilità

Né Turbo Basic né BASICA ha la funzione LTRIM$. In una versione di BASIC che non ha questa funzione, si può usare, per sostituirla, questa routine:

```
rtrim$ = st$
DO WHILE LEN(rtrimSt$) > 0 AND RIGHT$(rtrimSt$, 1) = " "
   rtrimSt$ = LEFT$(rtrimSt$, LEN(rtrimSt$) - 1)
LOOP
```

Questa routine copia il valore stringa memorizzato in *st$* e poi in *rtrimSt$* ed elimina gli spazi finali dalla copia.

SPACE$

La funzione SPACE$ ritorna una stringa di spazi.

```
SPACE(num)
```

L'argomento intero è un valore che va da 0 a 32.767 e specifica la lunghezza della stringa che SPACE$ creerà.

Un esempio di SPACE$

Un'istruzione PRINT nel programma *Mese* (presentato nel Capitolo 29) illustra l'utilizzo di SPACE$ per centrare una stringa all'interno di un'area orizzontale specificata:

```
PRINT SPACE$(12 - LEN(StrMe$) \ 2); StrMe$; anno%
```

Ecco l'argomento della funzione SPACE$ che è un conteggio basato sulla lunghezza della variabile stringa *StrMe$*.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione SPACE$.

STRING$

La funzione STRING$ ritorna una stringa costituita da un carattere ripetuto.

```
STRING$(num, cod)     ' Carattere ripetuto come specificato
                      ' dal numero di codice ASCII.

STRING$(num, str)     ' Il carattere ripetuto è il primo
                      ' carattere della str.
```

Il primo argomento di STRING$ è un intero che va da 0 a 32.767 e che specifica la lunghezza della stringa che la funzione produrrà. Il secondo argomento può presentarsi come un intero o come una stringa:

- Un argomento intero è un numero di codice ASCII (da 0 a 255) che rappresenta il carattere ripetuto.
- Il primo carattere in un argomento della stringa diventa il carattere ripetuto.

Un esempio di STRING$

Il programma *Agenda* (presentato nel Capitolo 30) usa la funzione STRING$ per creare una linea di trattini sotto un valore stringa visualizzato sullo schermo:

```
PRINT strData$
PRINT STRING$(LEN(strData$), "-")
```

In questo esempio, il primo argomento di STRING$ è la lunghezza del valore *strData$*. Il secondo argomento è un valore stringa literal: un trattino tra virgolette. (Vedere la voce LEN per ulteriori informazioni di questo esempio.) La funzione STRING$ produce una stringa costituita da un carattere selezionato. Per creare un modello ripetuto di più caratteri, si può scrivere una propria funzione; ad esempio, questa, chiamata *PiùStringhe$*, consente di ripetere più di un carattere:

```
FUNCTION PiùStringhe$ (ripet%, model$)

   CONST lunghMass% = 255
```

```
   FOR i% = 1 TO ripet%
      IF LEN(strOut$) < (lunghMass% - LEN(model$)) THEN
         strOut$ = strOut$ + model$
      END IF
   NEXT i%

   PiùStringhe$ = strOut$

END FUNCTION
```

Questa funzione accetta argomenti simili a quelli di STRING$, ad eccezione del secondo argomento che è un modello di caratteri da ripetere. Ecco un esempio di una chiamata alla funzione:

```
PRINT PiùStringhe$(10, "- * ")
```

Questa chiamata visualizza sullo schermo la seguente linea di caratteri:

```
- * - * - * - * - * - * - * - * - * - *
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione STING$.

UCASE$

La funzione UCASE$ ritorna l'equivalente maiuscolo di una stringa.

```
UCASE$(str)
```

L'argomento *str* può presentarsi come un valore literal messo tra virgolette, una variabile o un'espressione. Le lettere minuscole nell'argomento originale sono convertite in maiuscole nel valore di ritorno. Altri caratteri vengono copiati come si presentano.

Un esempio di UCASE$

Questo esempio è preso dal programma *Compleanno* (presentato nel Capitolo 28). Questo ciclo prende un solo carattere dall'utente tramite tastiera, lo

converte in maiuscolo e poi verifica la presenza del carattere in una stringa delle opzioni valide del menù:

```
DO
   Selez$ = UCASE$(INKEY$)
   Selez% = INSTR("NLADEC", selez$)
LOOP WHILE Selez$ = "" OR Selez% < 1
```

In effetti, la funzione UCASE$ in questo frammento consente all'utente di immettere l'input come una lettera maiuscola o minuscola.

Compatibilità

Turbo Basic ha la funzione UCASE$, ma BASICA no. In una versione di BASIC che non ha le funzioni di conversione in maiuscolo o minuscolo, si può usare questa routine per convertire tutte le lettere minuscole in maiuscole nella variabile *st$*:

```
diff% = ASC("a") - ASC("A")

FOR car% = 1 TO LEN(st$)
   car$ = MID$(st$, car%, 1)
   IF car$ >= "a" AND car$ <= "z" THEN
      car$ = CHR$(ASC(car$) + diff%)
      MID$(st$, car%, 1) = car$
   END IF
NEXT car%
```

Vedere le istruzioni ASC e MID$ per ulteriori approfondimenti sulla tecnica illustrata in questa routine.

Capitolo 20

Funzioni matematiche

Questo capitolo analizza una dozzina di funzioni matematiche che QuickBASIC fornisce per diversi scopi. Le funzioni sono divise in cinque differenti categorie:

- Funzioni trigonometriche
- Funzioni esponenziali e logaritmiche
- Funzioni per la conversione di base
- Funzioni per la generazione di numeri casuali
- Funzioni relative al segno

Alcune delle funzioni in queste categorie sono utili per applicazioni scientifiche o finanziarie che implicano l'esecuzione di specifiche operazioni matematiche. Altre compaiono spesso in spazi a sorpresa, come routine grafiche e programmi di giochi. Infatti, molte voci di questo capitolo analizzano esempi presi dal programma *DateStoriche*, il gioco presentato nel Capitolo 32. In particolare, il programma *DateStoriche* illustra l'uso delle funzioni trigonometriche per eseguire alcune operazioni grafiche collegate ai cerchi ed il generatore di numeri casuali QuickBASIC per controllare gli eventi casuali di un gioco.

FUNZIONI TRIGONOMETRICHE

QuickBASIC fornisce una serie di quattro funzioni trigonometriche: ATN (la funzione arcotangente), COS (la funzione coseno), SIN (la funzione seno) e TAN (la funzione tangente). A parte il loro significato nelle applicazioni scientifiche e tecniche, queste funzioni spesso si rivelano utili nei programmi grafici. Un esempio di programma di questo tipo viene discusso nelle voci COS e SIN.

ATN

La funzione ATN ritorna l'arcotangente del proprio argomento.

```
ATN(num)
```

L'argomento numerico è un valore positivo o negativo. ATN ritorna un angolo in radianti. Il risultato è un numero a precisione doppia o semplice, dipende dalla precisione dell'argomento dato. ATN è l'inverso di TAN. La funzione TAN ritorna la tangente dell'argomento di un angolo. ATN ritorna l'angolo corrispondente all'argomento della tangente.

Un esempio di ATN

Per valori grandi positivi o negativi, la funzione ATN ritorna un valore che si avvicina a + o - $\pi/2$, queste linee lo dimostrano:

```
gr# = 1000000000
PRINT USING "##,#### ##,####"; 2 * ATN(gr#); 2 * ATN(-gr#)
```

Il risultato di questa istruzione PRINT è

```
3,1416 -3,1416
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione ATN.

COS

La funzione COS ritorna il coseno di un angolo.

```
COS(num)
```

L'argomento numerico è un angolo positivo o negativo espresso in radianti. Il valore di ritorno, da -1 a 1, è un numero in precisione doppia o semplice, dipende dalla precisione dell'argomento dato.

Un esempio di COS

Il programma *DateStoriche*, presentato nel Capitolo 32, usa le funzioni COS e SIN per calcolare le posizioni della lancetta in un orologio ogni dieci secondi. (Il programma è un gioco a quiz ed utilizza l'orologio per misurare il tempo a disposizione per rispondere alla domanda corrente.) Il seguente frammento disegna ogni nuova posizione della lancetta:

```
LINE (0, 0)-(xCirc, yCirc), 15

ang% = ang% + incrAng%
xCirc = lunghLanc% * COS(Rad(ang%))
xCirc = lunghLanc% * SIN(Rad(ang%))
LINE (0, 0)-(xCirc, yCirc), 4
```

La prima istruzione LINE cancella la lancetta dalla sua precedente posizione sull'orologio. Dato un angolo incrementato ed un paio di istruzioni di assegnamento, si possono usare COS e SIN per calcolare le coordinate esterne *(xCirc, yCirc)* della nuova posizione della lancetta. Le formule generali per calcolare queste coordinate sono:

```
x = rag * COS(ang)
y = rag * SIN(ang)
```

Nel contesto del programma *DateStoriche*, moltiplicando il coseno per la diagonale della lancetta si ha la coordinata *x* e moltiplicando il seno si ha la coordinata *y*. Una volta che queste coordinate sono disponibili, un'altra istruzione LINE disegna la lancetta nella sua nuova posizione, iniziando da (0,[]0), cioè il centro dell'orologio:

```
LINE (0, 0)-(xCirc, yCirc), 4
```

Il programma *DateStoriche* contiene una funzione chiamata *Rad* che fornisce l'equivalente in radianti di un angolo in gradi. Ecco il codice della funzione:

```
FUNCTION Rad (inAng%)

   CONST PI = 3,14159
   Rad = (inAng% / 360!) * 2 * PI

END FUNCTION
```

Per definizione un cerchio completo di 360 gradi è equivalente a 2π radianti.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione COS.

SIN

La funzione SIN ritorna il seno di un angolo.

```
SIN(num)
```

L'argomento numerico è un angolo positivo o negativo espresso in radianti. Il valore di ritorno, da -1 a 1, è un numero in precisione doppia o semplice, dipende dalla precisione dell'argomento dato.

Un esempio di SIN

Il programma *DateStoriche*, presentato nel Capitolo 32, usa le funzioni SIN e COS per calcolare le posizioni della lancetta in un orologio ogni dieci secondi. (Il programma è un gioco a quiz ed utilizza l'orologio per misurare il tempo a disposizione per rispondere alla domanda corrente.) Il programma usa la funzione SIN per calcolare la coordinata *y* di ogni nuova posizione della lancetta:

```
xCirc = lunghLanc% * SIN(Rad(ang%))
```

Moltiplicando il seno si ha la coordinata *y*. (Vedere la voce COS per ulteriori dettagli su questo frammento. In particolare si noti che il programma *DateStoriche* contiene una funzione chiamata *Rad* che fornisce l'equivalente in radianti di un angolo in gradi.)

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione SIN.

TAN

La funzione TAN ritorna la tangente del suo argomento.

```
TAN(num)
```

L'argomento numerico è un angolo positivo o negativo espresso in radianti. Il valore di ritorno, da -1 a 1, è un numero in precisione doppia o semplice, dipende dalla precisione dell'argomento dato.

Un esempio di TAN

Il risultato della funzione tangente tende all'infinito per più-o-meno argomenti che tendono a π/2, eccone la dimostrazione:

```
DEFDBL P
pi = 4 * ATN(1)
PRINT USING "##,#^^^^ ##,#^^^^"; TAN(pi / 2); TAN(-pi / 2)
```

Il risultato di questa istruzione PRINT è:

```
1,6D+16 -1,6D+16
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione TAN.

FUNZIONI ESPONENZIALI E LOGARITMICHE

Le voci di questo capitolo descrivono tre funzioni matematiche che lavorano con esponenti e logaritmi:

- La funzione EXP ritorna una potenza di *e*, la base logaritmica dei numeri naturali uguale approssimativamente a 2,718282.
- La funzione LOG ritorna il logaritmo naturale in base *e*.
- La funzione SQR dà la radice quadrata di un numero.

Oltre a queste, si ricordi che QuickBASIC ha la funzione esponenziale (^), che dà il valore di un numero elevato a un esponente.

EXP

La funzione EXP ritorna il valore *e* alla potenza *x*, dove x è l'argomento di EXP.

```
EXP (num)
```

L'argomento numerico positivo o negativo può essere espresso come un valore literal, una variabile o un'espressione. EXP ritorna un valore in precisione semplice o doppia, dipende dal tipo di dato numerico dell'argomento. L'argomento massimo in precisione semplice è approssimativamente 88,7 e quello in precisione doppia è circa 709,78#. Gli argomenti più grandi segnalano errori run-time con un messaggio d'errore in "Traboccamento".

Un esempio di EXP

Il seguente esercizio visualizza semplicemente una selezione di valori in precisione semplice o doppia:

```
PRINT "Valori in precisione semplice EXP:"
PRINT
PRINT "e ="; EXP(1)
PRINT "Valore maggiore: "; EXP(88,72283)
PRINT
PRINT
PRINT "Valori in precisione doppia EXP:"
PRINT
PRINT "e ="; EXP(1#)
PRINT "Valore maggiore: "; EXP(709,782712893384#)
```

L'output di questo programma indica il valore naturale della costante *e* come un valore in precisione semplice e doppia, mostra inoltre i valori maggiori ammessi nei tipi di dati numerici in precisione semplice e doppia:

```
Valori in precisione semplice EXP:

e = 2,718282
Valori maggiori: 3,402799E+38

Valori in precisione doppia EXP:

e = 2,718281828459045
Valore maggiore: 1,797693134862273D+308
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione EXP. Inoltre Turbo Basic fornisce le funzioni EXP2 e EXP10, che ritornano rispettivamente le potenze di 2 e 10. In QuickBASIC in sostituzione di queste due funzioni, si può semplicemente usare l'operazione esponenziale (^).

LOG

La funzione LOG ritorna il logaritmo naturale del suo argomento.

```
LOG(num)
```

Il valore di ritorno della funzione logaritmica dei numeri naturali è l'esponente di *e* che dà l'argomento della funzione. L'argomento numerico deve essere un valore positivo e può essere espresso come una costante, una variabile o un'espressione. Un argomento negativo risulta in un errore run-time "Chiamata di funzione errata". LOG ritorna un valore in precisione semplice o doppia, dipende dal tipo di dato numerico dell'argomento.

Un esempio di LOG

Si può usare LOG per sviluppare una funzione che dia i logaritmi con qualsiasi base:

```
FUNCTION Logx (num, LogBase)

' La funzione Logx dà il logaritmo in base x di un
' numero. Il primo argomento è il numero ed il secondo è
' la base.

   Logx = LOG(num) / LOG(LogBase)

END FUNCTION
```

Questo esercizio esegue alcune semplici chiamate a questa funzione e a LOG stessa, per visualizzare i logaritmi in base *e*, 10 e 2.

```
PRINT LOG(EXP(1))
PRINT Logx(100, 10)
PRINT Logx(64, 2)

PRINT
x = 25
PRINT LOG(x)
PRINT Logx(x, 10)
PRINT Logx(x, 2)
```

Ecco l'output di queste istruzioni PRINT:

```
1
2
6
```

```
3,218876
1,39794
4,643856
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione LOG. Inoltre Turbo Basic fornisce le funzioni LOG2 e LOG10, per creare logaritmi in base 2 e 10. La funzione *Logx* listata qui sopra indica un modo per sostituire queste funzioni in QuickBASIC.

SQR

La funzione SQR ritorna la radice quadrata di un argomento numerico positivo.

```
SQR(num)
```

L'argomento numerico può comparire come un valore literal, una variabile o un'espressione. Il numero deve essere positivo, un valore negativo determina un errore run-time "Chiamata di funzione errata".

Un esempio SQR

Il seguente esercizio illustra la funzione SQR. Il programma trova e visualizza tutti i numeri da 1 a 900 che hanno radici quadrate intere:

```
CLS
n% = 0
FOR i% = 1 TO 900
   r = SQR(i%)
   IF r = INT(r) THEN
      PRINT USING "### "; i%;
      n% = n% + 1
      IF n% MOD 10 = 0 THEN PRINT
   END IF
NEXT i%
```

Ecco l'output del programma:

```
  1     4     9    16    25    36    49    64    81   100
121   144   169   196   225   256   289   324   361   400
441   484   529   576   625   676   729   784   841   900
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione SQR.

CONVERSIONI IN BASE NUMERICA

Per convenienza dei programmatori che vogliono lavorare con i sistemi in base numerica oltre che decimale, QuickBASIC fornisce due funzioni che danno gli equivalenti ottali ed esadecimali con base di 10 numeri. Queste funzioni, chiamate OCT$ e HEX$, ritornano valori stringa che contengono le cifre della base in questione:

- La funzione OCT$ dà l'equivalente ottale (base 8) di un argomento numerico. L'intervallo delle cifre ottali va da 0 a 7. Ogni posto in un numero ottale rappresenta una potenza di 8, ad esempio, 8^0, 8^1, 8^2, 8^3 e così via, da destra a sinistra.
- La funzione HEX$ fornisce l'equivalente esadecimale (base 16) di un argomento numerico. L'intervallo delle cifre esadecimali va da 0 a 9 e poi da A a F. Ogni posto in un numero esadecimale rappresenta una potenza di 16, ad esempio, 16^0, 16^1, 16^2, 16^3 e così via, da destra a sinistra.

Una cifra esadecimale è direttamente equivalente a quattro cifre binarie. Le cifre binarie sono 0 e 1. Ogni posto in un numero binario rappresenta una potenza di 2, ad esempio, 2^0, 2^1, 2^2, 2^3 e così via, da destra a sinistra. QuickBASIC non fornisce uno strumento che dà l'equivalente binario di un numero in base 10. Comunque, si può facilmente scrivere la funzione usando HEX$ come punto d'inizio. (Si può trovare questo strumento nella voce HEX$.)

HEX$

Dato un argomento numerico, la funzione HEX$ ritorna il valore equivalente esadecimale come una stringa di cifre.

```
HEX$(num)
```

L'argomento numerico può appartenere ad un qualsiasi tipo. (Se si fornisce un valore a virgola mobile come argomento, la funzione HEX$ prima arrotonda il valore in un intero o in un intero lungo.)

Un esempio HEX$

Il seguente programma, chiamato *ConversioneBase*, esegue tre diverse conversioni in base numerica per un valore numerico d'input:

```
' CONVERSIONEBASE.BAS
' Visualizza gli equivalenti binari, esadecimali e ottali
' di un intero che l'utente immette dalla tastiera.

DECLARE FUNCTION Bin$ (inVal&)

CLS
PRINT "Conversioni di Base"
PRINT
INPUT "Immettere un intero in base 10: ", inVal&
PRINT
PRINT "Binario:     "; Bin$(inVal&)
PRINT "Esadecimale: "; HEX$(inVal&)
PRINT "Ottale:      "; OCT$(inVal&)
PRINT

FUNCTION Bin$ (inVal&)

' La funzione Bin$ fornisce l'equivalente binario (in
' base 2) di un intero in base 10.

' Convertire l'argomento in esadecimale:

   e$ = HEX$(inVal&)
' Cifra per cifra, convertire il valore esadecimale in
' una sequenza di cifre binarie.
   outBin$ = ""
```

```
   FOR i% = 1 TO LEN(e$)

' Isolare una sola cifra esadecimale.

   eCar$ = MID$(e$, i%, 1)

' Stabilisce se la cifra è da "0" a "9" o da "A" a "F"

   eVal% = INSTR("ABCDEF", eCar$)
   IF eVal% <> 0 THEN
      eVal% = eVal% + 9
   ELSE
      eVal% = VAL(eCar$)
   END IF
' Trovare quattro cifre binarie che sono equivalenti ad
' una cifra esadecimale.

   valCor% = 0
   For j% = 3 TO 0 STEP -1
      IF 2 ^ j% + valCor% <= eVal% THEN
         outBin$ = outBin$ + "1"
         valCor% = valCor% + 2 ^ j%
      ELSE
         outBin$ = outBin$ + "0"
      END IF
   NEXT j%
NEXT i%

Bin$ = outBin$

END FUNCTION
```

Questo programma usa la funzione HEX$ in due diversi contesti. Per prima cosa, la sezione del programma principale chiama HEX$ per fornire l’equivalente esadecimale del valore d’input:

```
PRINT "Esadecimale: "; HEX$(inVal&)
```

Secondo, la funzione *Bin$* chiama HEX$ nel processo di determinazione dell’equivalente binario del valore d’input. Ecco un’attivazione tipo del programma:

```
Conversioni di Base

Immettere un intero in base 10: 45678
```

```
Binario:       1011001001101110
Esadecimale:   B26E
Ottale:        131156
```

Si noti che QuickBASIC offre le funzioni incorporate HEX$ e OCT$, ma non contiene nessuno strumento per ottenere l'equivalente binario di un numero. La funzione *Bin$* nel programma *ConversioneBase* suggerisce un semplice modo per creare una funzione di conversione binaria: prima si usa HEX$ per convertire un numero in esadecimale, poi viene convertito il valore esadecimale in binario, cifra per cifra.

Compatibilità

Sia Turbo Basic sia BASICA hanno le funzioni HEX$ e OCT$. Inoltre, Turbo Basic ha una funzione BIN$ incorporata.

OCT$

Dato un argomento numerico, la funzione OCT$ ritorna il valore equivalente ottale come una stringa di cifre.

```
OCT$(num)
```

L'argomento numerico può appartenere ad un qualsiasi tipo. (Se si fornisce un valore a virgola mobile come argomento, la funzione OCT$ prima arrotonda il valore in un intero o in un intero lungo.)

Un esempio OCT$

Il programma *ConversioneBase*, listato nella voce HEX$, contiene un esempio di chiamata ad una funzione OCT$:

```
PRINT "Ottale:   "; OCT$(inVal&)
```

Questa istruzione visualizza l'equivalente ottale di un valore d'input. (Vedere la voce HEX$ per ulteriori dettagli.)

Compatibilità

Sia Turbo Basic sia BASICA hanno le funzioni HEX$ e OCT$. Inoltre, Turbo Basic ha una funzione BIN$ incorporata.

NUMERI CASUALI

La funzione RND è il generatore di numeri casuali di QuickBASIC. Un'istruzione collegata, RANDOMIZE, è lo strumento per controllare l'esecuzione di RND in un dato programma. I numeri casuali sono importanti in molti tipi di applicazioni. Per prima cosa vengono in mente i programmi di giochi, perché gli eventi nei giochi si basano spesso sul caso. Comunque, il generatore QuickBASIC può essere utile anche in altri contesti di programmazione. Ad esempio, i programmatori talvolta usano i numeri casuali come dati d'input per verificare l'esecuzione del codice. Inoltre, i numeri casuali hanno un compito in molti tipi di programmi di simulazione: i programmi che tentano di simulare le condizioni, e prevedere i risultati, di operazioni tecniche, economiche o d'affari. L'esempio principale di utilizzo dei numeri casuali in questo libro è il programma *DateStoriche*, presentato nel Capitolo 32; si possono trovare frammenti del programma in entrambe le voci di questo paragrafo.

RANDOMIZE

L'istruzione RANDOMIZE assegna il punto d'inizio del generatore di numeri casuali di QuickBASIC.

```
RANDOMIZE         ' Richiede il numero casuale successivo.

RANDOMIZE num     ' Usa num come punto d'inizio.

RANDOMIZE TIMER   ' Usa TIMER per stabilire un punto d'inizio
                  ' imprevedibile.
```

RANDOMIZE controlla l'input numerico dalla funzione RND. Solitamente il compito di questa istruzione è garantire una sequenza imprevedibile di numeri dalle chiamate seguenti a RND. L'argomento numerico dell'istru-

zione RANDOMIZE è usato come un fattore "d'inizio" nel calcolare il numero successivo prodotto da RND. Per un dato valore "d'inizio", RND ritorna sempre la stessa sequenza di numeri. Di conseguenza, il modo migliore per creare una sequenza imprevedibile di numeri è iniziare con un numero casuale d'inizio. Di solito ciò viene eseguito chiamando la funzione TIMER per creare un numero d'inizio. TIMER ritorna il numero di secondi trascorsi dalla mezzanotte del giorno corrente; questo valore è diverso per ogni esecuzione di un programma che usa RANDOMIZE. Senza un argomento, RANDOMIZE visualizza un prompt sullo schermo, prendendo il numero d'inizio dall'utente tramite tastiera:

```
Numero d'inizio casuale (-32768 fino 32767)? -
```

Un esempio di RANDOMIZE

Il programma *DateStoriche*, presentato nel Capitolo 32, dà l'istruzione RANDOMIZE una volta vicino all'inizio di ogni esecuzione:

```
RANDOMIZE TIMER ' Attivare il generatore di numeri casuali.
```

Il programma usa la funzione RND per controllare gli eventi di un gioco a quiz. Come risultato di questa istruzione RANDOMIZE, ogni esecuzione del programma sarà diversa da quella precedente. (Vedere la voce RND per ulteriori dettagli.)

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione RANDOMIZE.

RND

La funzione RND ritorna un numero casuale che è maggiore o uguale a 0 e minore di 1.

```
RND      ' Ritorna il numero casuale successivo.
```

```
RND(pos)     ' Ritorna il numero casuale successivo.

RND(0)       ' Ritorna lo stesso numero casuale dato dalla
             ' precedente chiamata a RND.

RND(neg)     ' Ritorna sempre lo stesso numero casuale per
             ' un dato argomento negativo.
```

L'argomento numerico facoltativo di RND determina come la funzione ottiene il proprio risultato. Senza nessun argomento o con un argomento positivo, RND segue un algoritmo che crea il numero successivo in una data sequenza di numeri casuali. Dato un argomento pari a 0, una chiamata a RND ripete il numero della chiamata precedente. Infine, un dato argomento negativo crea sempre lo stesso numero casuale. L'istruzione RANDOMIZE inizializza il calcolo che RND impiega per creare una sequenza di numeri casuali. Durante lo sviluppo di un programma, l'istruzione RANDOMIZE insieme a varie altre opzioni di RND consentono di ripetere una data sequenza di numeri casuali in due diverse esecuzioni del programma.

Un esempio di RND

Il programma *DateStoriche*, presentato nel Capitolo 32, presenta un gioco a quiz che richiede al giocatore di mettere a confronto gli eventi storici con le loro date corrette. Il programma utilizza i numeri casuali per due scopi:

- Stabilire un ordine casuale per la sequenza di domande che il programma pone.
- Scegliere casualmente date errate per una lista di più risposte possibili.

In entrambi i casi, il programma richiede numeri casuali compresi in un intervallo di interi. Una funzione chiamata *InterCas%* usa RND per ottenere questi valori:

```
FUNCTION InterCas% (min%, max%)

' La funzione InterCas% ritorna un intero casuale tra
' min% e max%, compresi.

InterCas% = min% + INT(RND * (max% - min% + 1))

END FUNCTION
```

Data questa funzione, il seguente ciclo FOR riorganizza casualmente l'ordine delle domande nel vettore *infData*:

```
FOR i% = 1 TO numRec%
   j% = InterCas%(1, numRec%)
   SWAP infData(i%), infData(j%)
NEXT i%
```

Questa linea seleziona casualmente una data sbagliata come appartenente alla lista a più scelte:

```
listaData%(i%) = InterCas%(dataMin%, dataMax%)
```

(Il programma determina i valori appropriati per *dataMin%* e *dataMax%* così che ogni data sbagliata sarà compresa in un intervallo di anni attorno alla data effettivamente corretta.)

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione RND.

FUNZIONI RELATIVE AL SEGNO

Questa parte finale presenta due ulteriori funzioni numeriche:

- La funzione ABS ritorna il *valore assoluto*, o l'equivalente senza segno, di un numero.
- La funzione SGN ritorna un valore che indica il segno di un numero.

Mentre nessuna di queste funzioni è presente molto frequentemente nei programmi QuickBASIC, entrambe sono strumenti utili in alcune situazioni.

ABS

La funzione ABS ritorna il valore senza segno (positivo) del suo argomento numerico.

```
ABS(num)
```

L'argomento può appartenere ad un tipo numerico qualsiasi e può presentarsi come una costante, una variabile o un'espressione. Il valore di ritorno ha la stessa grandezza, o *valore assoluto*, dell'argomento, ma senza segno. In altre parole, ABS ritorna sempre un risultato positivo, senza tener conto del segno del suo argomento.

Un esempio di ABS

Un programma d'esempio discusso nelle voci GET e PUT (Capitolo 17, Parte IV) usa la funzione ABS per trovare le dimensioni di un'area rettangolare su uno schermo grafico:

```
deltaX% = ABS(x1% - x2%) + 1
deltaY% = ABS(y1% - y2%) + 1
```

Grazie alla funzione ABS, queste istruzioni danno interi positivi per *deltaX%* e *deltaY%*, senza tener conto delle posizioni relative degli indirizzi *(x1%, y1%)* e *(x2%, y2%)*.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione ABS.

SGN

La funzione SGN ritorna un valore pari a -1, 0 o 1, che rappresenta il segno del suo argomento numerico.

```
SGN(num)
```

L'argomento può appartenere ad un tipo numerico qualsiasi e può presentarsi come una costante, una variabile o un'espressione. Il valore di ritorno è -1 se l'argomento è negativo, 0 se l'argomento è zero o 1 se l'argomento è positivo.

Un esempio di SGN

Questo frammento di programma dimostra l'utilizzo di SGN per ottenere il formato di output:

```
SELECT CASE SGN(Tass)
   CASE -1
   form$ = "Bilancio tasse personale: ($$###,###.##)"
   CASE 0
   form$ = "Tasse pagate:    $#.## bilancio."
   CASE 1
   form$ = "Tasse pagate più del dovuto: $$###,###.##"

END SELECT

PRINT USING form$; Tass
```

Questo programma esamina il segno del valore numerico memorizzato in *Tass* e poi seleziona un formato d'output adatto tra le tre possibilità. Un'istruzione PRINT USING poi usa il formato selezionato per visualizzare il valore di *Tass*.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione SGN.

Capitolo 21

Conversioni di tipi di dati

Questo capitolo descrive molte funzioni incorporate che eseguono la conversione da un tipo di dati ad un altro. In questa parte si possono avere ulteriori notizie sui tre gruppi di funzioni:

- Funzioni che convertono tra loro i diversi tipi numerici: CDBL, CINT, CLNG e CSNG. Queste funzioni consentono di lavorare con successo con espressioni numeriche che contengono operandi di vari tipi di dati.
- Funzioni che ritornano valori interi partendo da argomenti in virgola mobile: CINT, FIX e INT. A seconda delle necessità del programma, si può scegliere una funzione che esegue questa particolare conversione in uno dei tanti modi possibili.
- Funzioni che convertono tra loro i tipi numerici e a stringa. Specificatamente, la funzione STR$ produce una stringa di cifre partendo da un argomento numerico e la funzione VAL ritorna un valore di tipo numerico partendo da una stringa di cifre.

Gli esempi presentati in questo capitolo sono progettati per illustrare questi strumenti e per mostrare perché le conversioni tra tipi di dati sono importanti.

CDBL

Dato un argomento numerico di un tipo qualsiasi, la funzione CDBL ritorna un valore equivalente che appartiene al tipo di dati DOUBLE (virgola mobile a doppia precisione).

```
CDBL(num)
```

L'argomento di CDBL può essere un valore numerico, una variabile o un'espressione compresa nell'intervallo ammesso per i valori di tipo a doppia precisione. (Il più grande valore in questo tipo numerico è 10 alla 308 e il più piccolo è 10 alla -324.) Il valore di ritorno è un numero che appartiene al tipo a doppia precisione.

Un esempio della funzione CDBL

Le conversioni di tipo vengono eseguite automaticamente in QuickBASIC quando si assegna un valore di un tipo numerico ad una variabile che appartiene ad un altro tipo. Ad esempio, questa istruzione memorizza il valore di *valLung&*, una variabile intera lunga, in *valDop#*, una variabile a doppia precisione:

```
valDop# = valLung&
```

In effetti, QuickBASIC converte una copia del valore memorizzato in *valLung&* in un valore a doppia precisione e memorizza questo valore convertito in *valDop#*. (Il valore memorizzato in *valLung&* non viene influenzato da questa operazione.) Comunque, in alcune operazioni aritmetiche intermedie, se una data operazione produce un valore esterno all'intervallo dei suoi singoli operandi, si ha un *traboccamento*. Ad esempio, si prenda in considerazione questo codice:

```
DEFLNG L
DEFDBL D

valLung1 = 778655989
valLung2 = 113466691
```

```
valDop = valLung1 * valLung2    ' Qui si ha un traboccamento.
PRINT valDop
```

In questo frammento, il tentativo di moltiplicare *valLung1* per *valLung2* dà un errore run-time *(traboccamento)*, perché il prodotto di questi due interi lunghi è esterno all'intervallo del tipo di dati LONG. È possibile prevenire il traboccamento convertendo uno degli operandi in un valore a doppia precisione, prima che si esegua la moltiplicazione. Questo è il compito della funzione CDBL:

```
valDop = CDBL(valLung1) * valLung2
```

Grazie a questa conversione, QuickBASIC esegue la moltiplicazione come un prodotto di due valori a doppia precisione e poi memorizza con successo il risultato nella variabile a doppia precisione *valDop*.

Compatibilità

Sia BASICA che Turbo Basic supportano la funzione CDBL.

CINT

Dato un argomento numerico in virgola mobile, la funzione CINT ritorna un valore intero arrotondato.

```
CINT(num)
```

L'argomento può presentarsi come un valore numerico literal, una variabile o un'espressione, ma deve essere compreso all'interno dell'intervallo di interi che va da -32.768 a 32.767. L'arrotondamento viene sempre eseguito se la prima cifra dopo la virgola è maggiore di 5. Dato un argomento positivo, CINT lo arrotonda all'intero superiore, dato un numero negativo CINT lo arrotonda all'intero inferiore. Ad esempio queste espressioni danno rispettivamente i valori 10 e -10:

```
CINT(9,6)
CINT(-9,6)
```

Se la parte decimale dell'argomento è esattamente 5, CINT o arrotonda o tronca per ottenere un risultato intero *pari*. Per esempio entrambe queste espressioni danno come risultato 10:

```
CINT(9,5)
CINT(10,5)
```

Se la parte decimale è minore di 5, CINT semplicemente tronca il valore per ottenere un intero.

Un esempio della funzione CINT

Questa istruzione, presa dal programma *Compleanno*, presentato nel Capitolo 28, usa la funzione CINT per arrotondare il risultato del calcolo dell'età all'anno più vicino:

```
CalcEtà% = CINT((annoCorr& - dataNasc&) / 365)
```

Questa linea è presa da una funzione chiamata *CalcEtà%*, il cui compito è quello di calcolare l'età di un dipendente nel giorno del suo compleanno.

Compatibilità

La funzione CINT di Turbo Basic si comporta come quella di QuickBASIC, mentre in BASICA differisce in un dettaglio: se la parte decimale di un argomento è esattamente 5, CINT arrotonda sempre all'intero successivo, non ottenendo necessariamente un risultato pari. Ad esempio, queste due espressioni danno, in BASICA, come risultato rispettivamente 10 e 11:

```
CINT(9,5)
CINT(10,5)
```

CLNG

Dato un argomento numerico, la funzione CLNG ritorna un valore equivalente che appartiene al tipo di dati LONG (intero lungo).

```
CLNG(num)
```

L'argomento di CLNG può essere un valore numerico, una variabile o un'espressione all'interno dell'intervallo ammesso per i valori di tipo intero lungo. (Questo intervallo va da -2.147.483.648 a 2.147.483.647.) Il valore di ritorno è un numero che appartiene al tipo intero lungo.

Un esempio della funzione CLNG

In alcune operazioni aritmetiche intermedie, se una data operazione dà un valore esterno all'intervallo dei suoi singoli operandi, si ha un *traboccamento*. (Vedere la voce CDBL per ulteriori approfondimenti sulle conversioni di tipi di dati e sul traboccamento.) Ad esempio, in questo frammento il tentativo di moltiplicare *valInt1* per *valInt2* dà un errore run-time (*traboccamento*), perché il prodotto di questi due interi è esterno all'intervallo del tipo di dati INTEGER:

```
DEFLNG L
DEFINT I

valInt1 = 1234
valInt2 = 5678

valLung = valInt1 * valInt2     ' Qui avviene un traboccamento.
PRINT valLung
```

Si può prevenire il traboccamento convertendo uno degli operandi in un valore intero lungo, prima di eseguire la moltiplicazione. Questo è un compito della funzione CLNG:

```
valLung = CLNG(valInt1) * valInt2
```

Grazie a questa conversione QuickBASIC esegue la moltiplicazione come il prodotto di due valori interi lunghi e memorizza con successo il risultato in una variabile intera lunga *valLung*.

Compatibilità

Turbo Basic supporta la funzione CLNG. Gli interi lunghi non sono definiti in BASICA.

CSNG

Dato un argomento numerico, la funzione CSNG ritorna un valore equivalente che appartiene al tipo di dati SINGLE (virgola mobile a precisione semplice).

```
CSNG(num)
```

L'argomento di CSNG può essere un valore numerico, una variabile o un'espressione all'interno dell'intervallo possibile dei valori di tipo intero lungo. (Il massimo valore possibile in questo tipo numerico è 10 elevato a 38 e il minimo è 10 alla -45.) Il valore di ritorno è un numero che appartiene al tipo a precisione semplice.

Un esempio della funzione CSNG

In alcune operazioni aritmetiche intermedie, se una data operazione dà un valore esterno all'intervallo dei suoi singoli operandi, si ha un *traboccamento*. (Vedere la voce CDBL per ulteriori approfondimenti sulle conversioni di tipi di dati e sul traboccamento.) Ad esempio, il seguente tentativo di moltiplicare *valInt1* per *valInt2* e assegnare il risultato alla variabile a precisione semplice *valSemp* dà un errore run-time (*traboccamento*), perché il prodotto di questi due interi è esterno all'intervallo del tipo di dati INTEGER:

```
DEFSNG S
DEFINT I

valInt1 = 1234
valInt2 = 5678

valSemp = valInt1 * valInt2    ' Qui avviene un traboccamento.
PRINT valSemp
```

Si può prevenire il traboccamento convertendo uno degli operandi in un valore intero lungo, prima di eseguire la moltiplicazione. Questo è un compito della funzione CSNG:

```
valSemp = CSNG(valInt1) * valInt2
```

Grazie a questa conversione QuickBASIC esegue la moltiplicazione come il prodotto di due valori interi lunghi e memorizza con successo il risultato in una variabile intera lunga *valSemp*.

Compatibilità

Sia Turbo Basic sia BASICA supportano la funzione CSNG.

FIX

Dato un argomento in virgola mobile, la funzione FIX tronca la parte decimale del numero e ritorna la parte intera non modificata.

```
FIX(num)
```

L'argomento di FIX può essere un valore numerico literal, una variabile o un'espressione. (L'argomento numerico non deve necessariamente essere compreso in un intervallo di interi o interi lunghi pari, sebbene l'uso appropriato di FIX sia normalmente nel contesto di uno di questi due intervalli.)

Un esempio della funzione FIX

Per un numero positivo, FIX dà un valore intero che è minore o uguale all'argomento; equivale quindi a INT. Ad esempio, entrambe queste espressioni ritornano 25:

```
INT(25,89)
FIX(25,89)
```

Per un numero negativo, FIX ritorna un valore che è maggiore o uguale all'argomento. Ad esempio, entrambe queste espressioni ritornano -25:

```
FIX(-25,39)
FIX(-25,88)
```

FIX non è uguale a INT per i numeri negativi. Le espressioni INT(-25,39) e INT(-25,88) danno -26.

Compatibilità

La funzione FIX è disponibile sia in BASICA sia in Turbo Basic. Turbo Basic ha anche una funzione intera supplementare, chiamata CIEL. Per i numeri positivi e negativi che contengono parti decimali diverse da zero, la funzione CIEL arrotonda sempre all'intero più vicino. Ad esempio, le seguenti espressioni ritornano rispettivamente -1 e 3:

```
CEIL(-1,75)
CEIL(2,25)
```

Per gli argomenti negativi, CEIL si comporta come FIX. Questa funzione QuickBASIC è un modo per implementare una versione della funzione CEIL di Turbo Basic:

```
FUNCTION Ceil% (num)
   IF num <= 0 THEN
      Ceil% = FIX(num)
   ELSE
      Ceil% = ABS(INT(-1 * num))
   END IF
END FUNCTION
```

INT

Dato un argomento in virgola mobile, la funzione INT elimina la parte decimale del numero e ritorna l'intero più grande che è minore o uguale all'argomento.

```
INT(num)
```

L'argomento di INT può essere un valore numerico literal, una variabile o un'espressione. (L'argomento numerico non è necessariamente compreso nell'intervallo di interi o interi lunghi pari, sebbene l' uso appropriato di INT sia normalmente nel contesto di uno di questi due intervalli.) Per i valori

positivi o negativi, il risultato di INT è sempre minore o uguale all'argomento. Ad esempio, queste due espressioni danno rispettivamente 9 e -10:

```
INT(9,2)
INT(-9,2)
```

Un esempio della funzione INT

Il programma *Compleanno*, presentato nel Capitolo 28, calcola le gratifiche annuali per il compleanno degli impiegati di una piccola ditta. Le gratifiche vengono calcolate moltiplicando L.50.000 per il numero degli anni di servizio di ogni impiegato. Uno dei campi nel database del dipendente, *anniServ*, è un valore a precisione semplice che rappresenta, per ogni dipendente, gli anni di lavoro nella ditta fino all'anno corrente. Questo valore comprende una parte decimale, ad esempio 4,8. Dato il valore del campo, questa linea calcola la gratifica per ogni impiegato:

```
grat% = INT(staff(i%).anniServ) * BONUSPERYEAR
```

La cifra da moltiplicare, L.50.000, viene rappresentata nel programma dalla costante simbolica BONUSPERYEAR. La funzione INT porta una delle specificazioni del conteggio della gratifica, cioè un dipendente riceve L.50.000 per ogni anno *intero* lavorato.

Compatibilità

La funzione INT si comporta allo stesso modo sia in BASICA sia in Turbo Basic.

STR$

Dato un argomento, la funzione STR$ ritorna una rappresentazione in forma di stringa del numero.

```
STR$(num)
```

L'argomento di STR$ può essere un valore numerico literal, una variabile o un'espressione. Il valore di ritorno è una stringa di cifre, preceduta dal segno meno se il numero è negativo oppure da uno spazio se il numero è positivo. (Si può utilizzare la funzione LTRIM$ per eliminare lo spazio iniziale dal risultato di STR$.) Se si fornisce un argomento nel formato ottale (&O) o esadecimale (&H), STR$ converte il valore di ritorno nella sua forma decimale. Ad esempio, questa chiamata a STR$ ritorna il valore stringa "65535":

```
STR$(&HFFFF&)
```

(Per ulteriori informazioni sul formato ottale ed esadecimale, consultare il Capitolo1.)

Alcuni esempi della funzione STR$

Il programma *Agenda*, presentato nel Capitolo 30, usa una funzione chiamata *DataCompl$* per produrre una stringa di dati dai tre argomenti interi. Ecco il codice della funzione:

```
FUNCTION DataCompl$ (mese%, giorno%, anno%)

   gio% = NumGior%(mese%, giorno%, anno%)
   strData = giorSett$(gio%) + " ., " + nomiMese$(mese%)

   IF mese% <> 5 THEN strData$ = strData$ + "."
   strData$ = strData$ + STR$(giorno%) + "," + STR$(anno%)

   DataCompl$ = strData$

END FUNCTION
```

I tre argomenti rappresentano il mese, giorno e anno di una data in questione. La funzione crea la corrispondente stringa della data, passo per passo, memorizzando le varie parti della stringa nella variabile *strData$*. Per prima cosa, la funzione stabilisce i nomi dei giorni della settimana ed il mese; poi, utilizzando la funzione STR$, la routine trova gli equivalenti stringa del giorno e gli interi dell'anno e concatena questi valori nella stringa.

Compatibilità

Sia BASICA sia Turbo Basic supportano la funzione STR$.

VAL

Dati una stringa di cifre ed il suo argomento, la funzione VAL ritorna un valore di tipo numerico equivalente.

```
VAL(str)
```

Si può esprimere l'argomento di VAL come una stringa literal tra virgolette, una variabile stringa o un'espressione stringa. VAL legge questi caratteri come parte di un numero:

- Le cifre, da "0" a "9"
- I segni aritmetici, "+" o "–"
- Un punto decimale, "."
- Le lettere "E" o "D", maiuscolo o minuscolo, per un numero espresso in formato esponenziale a precisione semplice o doppia (ad esempio, "1,5E8")
- La notazione di prefisso "&H" per un numero esadecimale e le cifre esadecimali da "0" a "9" e da "A" a "F"
- La notazione di prefisso "&O" per un numero ottale e le cifre ottali da "0" fino a "7".

Se VAL incontra un carattere che non può essere letto come parte di un numero, la funzione ignora il resto della stringa. VAL elimina gli spazi vuoti dall'inizio della stringa, altrimenti, se il primo carattere (diverso da spazio) non è parte di un numero, il valore di ritorno è zero.

Alcuni esempi della funzione VAL

Il programma *Compleanno*, presentato nel Capitolo 28, contiene molti esempi della funzione VAL, compresa questa istruzione:

```
an% = VAL(RIGHT$(DATE$, 4))
```

Questo assegnamento memorizza un intero a quattro cifre dell'anno nella variabile *an%*. La funzione DATE$ fornisce una stringa a dieci caratteri che rappresenta la data corrente (ad esempio, "06/07/1989"). La funzione RIGHT$ in questa espressione ritorna gli ultimi quattro caratteri della stringa della data. Infine, la funzione VAL converte queste cifre in un valore di tipo intero. Il programma *Agenda*, discusso nel Capitolo 30, comprende molti interessanti utilizzi di VAL. Il programma visualizza sullo schermo l'agenda per una data selezionata e poi usa un dialogo d'input per aggiungere nuove informazioni all'agenda del giorno. Uno dei primi compiti del programma è leggere le istruzioni della linea di comando dell'utente per selezionare una data; il programma aspetta la data, fornita dall'utente dal prompt del DOS che deve essere nel formato "mm/gg/aa" o "mm/gg/aaaa". Questo frammento, preso da un sottoprogramma chiamato *LeggiInData*, usa la funzione VAL per convertire i tre elementi di questa data immessa in valori interi:

```
posSlash1% = INSTR(inData$, slash)
posSlash2% = INSTR(posSlash1% + 1, inData$, slash)

IF posSlash1% <> 0 THEN
   mese% = VAL(inData$)
   giorno% = VAL(MID$(inData$, posSlash1% + 1))

IF posSlash2% <> 0 THEN
   anno% = VAL(MID$(inData$, posSlash2% + 1))
```

Questa routine inizia usando la funzione INSTR per trovare le posizioni dei due caratteri slash che separano parti della data nella stringa *inData$*. Per determinare i valori interi *mese%* e *giorno%*, il programma si basa sul fatto che la funzione VAL smette di leggere la stringa quando incontra un carattere non numerico. Ad esempio, l'espressione *VAL(inData$)* legge le cifre solo fino al primo carattere slash in *inData$* e ritorna l'intero equivalente di queste cifre. Un'altra routine, presa dal sottoprogramma *Corr* nel programma *Agenda* legge le istruzioni dell'utente per rivedere l'agenda del giorno come è visualizzata su schermo. L'utente può scegliere un'ora del mattino (dalle 7 alle 12) o del pomeriggio (dalla 1 alle 6) e poi immettere una nuova linea per quell'ora, oppure l'utente può inserire la Q per terminare il dialogo d'input:

```
PRINT "Immettere l'ora (<7> alle <12> o <1> alle <6>) "
INPUT "per un nuovo appuntamento, o immettere <Q> per uscire: ", inVal$
PRINT
IF INSTR(UCASE$(inVal$), "Q") <> 0 THEN
   eseg% = TRUE
ELSE
   inVal% = VAL(inVal$)
IF inVal% >= 1 AND inVal% <= 12 THEN salvCorr% = TRUE

SELECT CASE inVal%
' ...
```

Si noti che l'input dell'utente è accettato come un valore stringa e memorizzato nella variabile *inVal$*. La routine prima cerca una "Q" in *inVal$* per vedere se l'utente vuole uscire. Se non ci sono "Q" nella stringa, il programma allora usa VAL per convertire *inVal$* nell'intero *inVal%*. Se *inVal%* è un intero compreso tra 1 e 12, il programma continua il dialogo d'input. Convenientemente, la funzione VAL consente a questa routine di leggere l'input dell'utente o come una stringa o come un numero.

Compatibilità

Sia BASICA sia Turbo Basic supportano la funzione VAL.

Parte VI

Gestione degli errori e degli eventi

La Parte VI analizza due tecniche speciali chiamate gestione degli errori e degli eventi e presenta le potenti istruzioni QuickBASIC che rendono possibili queste tecniche. Il compito di una routine della gestione errori o eventi è interrompere temporaneamente il normale corso di un'operazione del programma in risposta ad un evento specifico:

- Una routine per la *gestione degli errori* anticipa un potenziale errore run-time. Se l'errore avviene mentre la gestione è attivata, il controllo del programma passa ad una *routine di gestione degli errori*. Idealmente, questa routine risolve il problema che ha causato l'errore e evita un indesiderabile interruzione nell'esecuzione del programma.
- Una routine per la *gestione degli eventi* è una routine simile che prevede un evento specifico e risponde in modo appropriato. L'evento può richiedere tante altre attività esterne: alcuni tipi di input da tastiera, l'input da un altro dispositivo hardware, il trascorrere del tempo misurato dall'orologio del sistema o l'attivazione di qualche altro evento. Se un evento avviene mentre la routine di gestione degli eventi è attiva, il flusso di controllo passa ad un *sottoprogramma per la gestione degli eventi* progettato per rispondere all'evento.

La gestione degli errori, le istruzioni e le funzioni QuickBASIC che si usano per creare questa gestione vengono trattate nel Capitolo 22. Molti programmi presentati in questo libro utilizzano le tecniche di gestione degli errori,

in particolare, il programma *Agenda* (nel Capitolo 30, Parte VIII) ed il programma *Linee* (presentato come parte della voce EOF nel Capitolo 11, Parte III). Nel Capitolo 22 si possono trovare frammenti di questi ed altri programmi.

Il Capitolo 23 tratta l'argomento della gestione degli eventi: per prima cosa viene discussa la gestione degli eventi generati dalla tastiera come un argomento a se stante e poi vengono riassunte tutte le altre forme di gestione degli eventi. Il programma *GestioneDate* presentato nel Capitolo 32, Parte VIII, illustra le tecniche e gli usi della gestione degli eventi. *GestioneDate* è progettato specificatamente per migliorare le prestazioni del programma *Agenda*.

La gestione degli errori di QuickBASIC e le tecniche per la gestione degli eventi utilizzano diramazioni del controllo stile GOTO e GOSUB, non le chiamate alle procedure SUB o FUNCTION. Di regola, in QuickBASIC, l'utilizzo delle istruzioni di GOTO e GOSUB è considerato un anacronismo stilistico. Comunque, le attività di gestione degli errori e degli eventi rappresentano delle eccezioni a questa regola. Come si può vedere negli esempi della Parte VI, le diramazioni GOTO ed i sottoprogrammi stile GOSUB sono parti importanti dei programmi che definiscono la gestione degli errori e degli eventi.

Capitolo 22

Gestione degli errori

L'istruzione per la gestione degli errori è lo strumento principale utilizzato per creare una gestione degli errori da programma e per identificare la routine che assumerà il controllo se effettivamente ci sarà un errore. Oltre a questa istruzione, QuickBASIC fornisce molti strumenti che sono importanti nel progetto di una routine di gestione degli errori: funzioni come ERR e ERDEV$ forniscono informazioni specifiche sugli eventi che hanno attivato la gestione d'errore, e l'istruzione RESUME rinvia il controllo al programma principale dopo che la routine di gestione degli errori ha terminato il proprio compito. Questo capitolo descrive tutte queste istruzioni e funzioni e i relativi strumenti di programmazione.

END

L'istruzione END segna la fine di un programma. Nei programmi che contengono gestioni di errori e di eventi, END separa il programma principale dalle routine di gestione degli eventi e degli errori.

```
END
```

END chiude i file aperti e termina l'esecuzione in corso del programma. END è opzionale in molti programmi: infatti QuickBASIC esegue un'implicita END dopo l'ultima istruzione del programma principale. Nessuna istruzione END viene richiesta per separare il programma principale dalle

procedure SUB e FUNCTION. Comunque, nella gestione degli errori e degli eventi, l'istruzione END è essenziale per dividere il programma principale dalle routine di gestione degli errori e degli eventi. Questo tipo di routine, identificata da un'etichetta di linea, va eseguita solo sotto il controllo di un'istruzione *event* ON o ON ERROR. Ma una routine della gestione degli eventi o degli errori potrebbe essere eseguita involontariamente se il programma non contenesse nessuna istruzione END per separare il programma principale dal codice della routine stessa.

Alcuni esempi di END

Questo codice illustra l'uso di END in un programma che stabilisce una gestione degli errori:

```
CLS
ON ERROR GOTO ProbDisc
   FILES "A:"
   PRINT
   FILES "B:"
ON ERROR GOTO 0

END

ProbDisc:
   PRINT "Non si trova il disco nel ";
   PRINT "drive "; ERDEV$;
   PRINT " (disp #"; ERDEV; ")"
RESUME NEXT
```

La routine di gestione degli errori che si trova sotto l'etichetta *ProbDisc* viene eseguita solo se attivata da un errore run-time. L'istruzione END impedisce un'esecuzione involontaria della routine. L'istruzione END può anche essere eseguita volontariamente, come in questo esempio:

```
IF noTrov% THEN END
```

Questa istruzione è presa da un programma chiamato *Linee*, discussa nella voce EOF (Capitolo 11, Parte III). Il programma cerca di aprire un file per leggerlo, se il file non è disponibile, la gestione degli errori assegna un valore logico vero alla variabile *noTrov%*. L'istruzione IF poi usa END per terminare l'esecuzione.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione END.

ERDEV E ERDEV$

Le funzioni ERDEV e ERDEV$ identificano il dispositivo hardware responsabile di un precedente errore run-time.

```
ERDEV
ERDEV$
```

Nessuna funzione ha un argomento. ERDEV ritorna un numero intero di codice che rappresenta l'errore e ERDEV$ ritorna il nome del dispositivo stesso.

Alcuni esempi di ERDEV e ERDEV$

Il seguente programma usa l'istruzione FILES per visualizzare gli indirizzari dei dischi nei drive A e B. Una gestione degli errori dispone la visualizzazione di un appropriato messaggio se un errore avviene durante il tentativo di accesso ad un disco. Un errore di questo tipo potrebbe verificarsi se lo sportellino del drive è aperto o se il drive non contiene nessun disco:

```
CLS
ON ERROR GOTO ProbDisc
   FILES "A:"
   PRINT
   FILES "B:"
ON ERROR GOTO 0

END

ProbDisc:
   PRINT "Non si trova il disco nel ";
   PRINT "drive "; ERDEV$;
   PRINT " (disp #"; ERDEV; ")"
RESUME NEXT
```

La routine di gestione degli errori che si trova sotto l'etichetta *ProbDisc* usa le funzioni ERDEV e ERDEV$ per identificare il drive che ha causato l'errore. Ad esempio, in questa esecuzione viene visualizzato un indirizzario per il drive A, mentre il drive B sembra essere aperto o vuoto:

```
A:\
SCHEDULE.BAS   TRAPDATE.BAS   BIRTHDAY.TXT   MONTH    .BAS
SCHEDULE.DAT   SCHEDULE.NDX   PARTX    .WS   NEWCHRON.TXT
72704 Byte liberi

Non si trova il disco nel drive B: (disp # 2)
```

Il messaggio "Non si trova il disco nel drive B: (disp # 2)" è fornito dalla routine gestione degli errori quando il programma cerca, senza riuscirvi, di accedere al disco nel drive B.

Compatibilità

Turbo Basic e le versioni recenti di BASIC Microsoft hanno le funzioni ERDEV e ERDEV$. Le prime versioni di BASICA, invece, non l'avevano.

ERL E ERR

Le funzioni ERL e ERR identificano un errore run-time che è stato gestito come un'istruzione ON ERROR.

```
ERL
ERR
```

Nessuna funzione ha un argomento, ma entrambe ritornano valori interi. ERL ritorna il numero di linea dell'istruzione che causa l'errore. (In un programma che non contiene numeri di linea, ERL ritorna sempre un valore pari a 0.) ERR ritorna il numero di codice di errore di QuickBASIC che corrisponde all'errore verificatosi.

Un esempio di ERR

Usando la funzione ERR per identificare l'errore verificatosi, una routine di gestione degli errori è in grado di ripristinare il funzionamento appropriato dell'operazione. Ad esempio, il programma *Agenda* (listato nel Capitolo 30) crea una gestione degli errori che viene attivata se sul disco non si trova il file SCHEDULE.NDX:

```
ON ERROR GOTO noIndFile
   OPEN "AGENDA.NDX" FOR INPUT AS #2
ON ERROR GOTO 0
```

Se si verifica un errore mentre il programma cerca di aprire il file, il controllo passa alla routine che si trova sotto l'etichetta *noIndFile:*

```
noIndFile:
   IF ERR = fileNoTrov THEN
      CLS
      PRINT "Errore nel file:"
      PRINT "---------"
      PRINT "AGENDA.DAT (il file del database agenda) esiste,"
      PRINT "ma il file ad indice, AGENDA.NDX, non si trova."
      PRINT "Il programma Agenda non può continuare."
      PRINT
      PRINT "Premere un tasto per terminare il programma."
      SLEEP
      END
   ELSE
      ON ERROR GOTO 0
   END IF
```

Questa routine è progettata specificatamente per presentare un messaggio d'errore sullo schermo e per terminare poi l'esecuzione del programma, se la ricerca su disco di AGENDA.NDX non ha successo. L'errore "File non trovato", con un codice d'errore 53, è rappresentato nel programma con la costante simbolica *fileNoTrov*:

```
CONST fileNoTrov = 53
```

Se la funzione ERR conferma che questo errore è avvenuto, la routine di gestione degli errori reagisce di conseguenza:

```
IF ERR = fileNoTrov THEN
```

Comunque, se si verifica qualche altro errore, la routine usa l'istruzione ON ERROR GOTO 0 per terminare il programma:

```
ELSE
   ON ERROR GOTO 0
END IF
```

All'interno di una routine di gestione degli errori, ON ERROR GOTO 0 termina il programma e fa sì che l'ambiente di QuickBASIC visualizzi un appropriato messaggio d'errore sullo schermo. (Vedere la voce ON ERROR per ulteriori informazioni.) Riassumendo, questa routine utilizza ERR per scegliere tra due operazioni, una progettata per un particolare errore e l'altra per un altro errore che potrebbe verificarsi.

Compatibilità

Sia Turbo Basic sia BASICA hanno le funzioni ERR e ERL.

ERROR

L'istruzione ERROR è uno strumento per il debugging dei programmi che attiva una gestione degli errori simulando un particolare errore.

```
ERROR cod
```

L'argomento *cod* è un numero del codice di errore QuickBASIC, un intero che va da 1 a 255. ERROR fa sì che QuickBASIC si comporti come se l'errore rappresentato da *cod* fosse effettivamente avvenuto.

Un esempio di ERROR

Il migliore utilizzo di ERROR è per verificare il comportamento di una routine di gestione degli errori commessi. (Una volta che il processo di verifica è completo, si può normalmente eliminare l'istruzione ERROR dal programma.) Ad esempio, il programma *Agenda*, listato nel Capitolo 30, stabilisce la seguente gestione d'errore nella possibilità della scomparsa di un file:

```
ON ERROR GOTO noIndFile
   OPEN "AGENDA.NDX" FOR INPUT AS #2
ON ERROR GOTO 0
```

La routine collocata sotto l'etichetta *noIndFile* esegue alcune operazioni se il file AGENDA.NDX non viene trovato nel disco. (Vedere la voce ERR per ulteriori dettagli.) Per attivare la gestione e verifica del risultato della routine di gestione degli errori è possibile riscrivere così questo frammento temporaneamente:

```
ON ERROR GOTO noIndFile
   ERROR 53
   ' OPEN "AGENDA.NDX" FOR INPUT AS #2
ON ERROR GOTO 0
```

Il numero di codice 53 rappresenta l'errore "File non trovato". Come risultato di questa istruzione ERROR, la gestione degli errori è attivata ed il controllo passa alla routine di gestione degli errori che si trova in *noIndFile*. (È possibile anche cercare altri numeri ERROR per vedere come la routine reagisce agli errori oltre a quello specifico progettato per la gestione.) Una volta che si è verificata la routine di gestione degli errori, bisogna eliminare l'istruzione ERROR e rimemorizzare l'istruzione OPEN nel suo originale formato.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione ERROR.

ON ERROR

L'istruzione ON ERROR crea una gestione degli errori ed identifica la corrispondente routine di gestione degli errori.

```
ON ERROR GOTO num    ' num è la posizione della routine
                     ' di gestione degli errori.

ON ERROR GOTO etic   ' etic è la posizione della routine
                     ' di gestione degli errori.

ON ERROR GOTO 0      ' Disattiva la gestione degli errori.
```

Una gestione degli errori è progettata per gestire potenziali errori run-time che si verificano durante un'esecuzione del programma. Si usa un'istruzione iniziale ON ERROR per attivare la gestione degli errori all'inizio di un blocco del codice in cui si anticipa un possibile errore. Il numero (o l'etichetta) della linea nell'istruzione ON ERROR specifica la posizione della *routine di gestione degli errori*. In base al tipo di errore che ci si aspetta, si può progettare questa routine per eseguire una delle seguenti operazioni:

- Correggere, consentendo al programma di continuare nonostante l'errore commesso.
- Creare una variabile globale che segnalerà l'errore al resto del programma.
- Identificare l'errore che si è verificato effettivamente, tra una serie di possibili errori.
- Visualizzare sullo schermo un messaggio d'errore e/o terminare l'esecuzione del programma.

In genere, si vuole che una gestione degli errori sia attiva solo per il blocco specifico di codice in cui ci si aspetta che avvenga un particolare errore run-time. Si segnala l'inizio di questo blocco con un'istruzione ON ERROR che identifica la routine di gestione degli errori. Alla fine del blocco si mette un'istruzione ON ERROR GOTO 0 che disattiva la gestione nel punto in cui non ci si aspetta più che avvenga l'errore. All'interno di una routine di gestione degli errori, ON ERROR GOTO 0 serve per diversi scopi: termina il programma e fà sì che l'ambiente di QuickBASIC visualizzi il messaggio d'errore run-time che sarebbe apparso sullo schermo se non si fosse verificata alcuna gestione degli errori. Un programma può contenere più gestioni di errore per potenziali errori in diversi contesti; può essere attivata, però, una sola gestione per volta. Una routine di gestione degli errori può essere in un sottoprogramma (definito con un'istruzione SUB) o in una funzione (definita con un'istruzione FUNCTION). In altre parole, il numero o l'etichetta della linea che segna l'inizio della routine deve essere collocato nel programma principale.

Alcuni esempi di ON ERROR

Il programma *Linee*, listato e discusso nella voce EOF (Capitolo 11, Parte III) contiene un semplice esempio di una gestione degli errori. Il programma

è progettato per contare il numero di linee non vuote in un file di testo. L'utente dà un nome al file in questione dalla linea di comando del DOS ed il programma utilizza la funzione COMMAND$ per leggere il nome:

```
nomeFile$ = COMMAND$
```

Il passo successivo è aprire il file per una lettura sequenziale. Comunque la gestione degli errori anticipa la possibilità che l'utente abbia dato il nome di un file inesistente:

```
ON ERROR GOTO noQueFile
   OPEN nomeFile$ FOR INPUT AS #1
ON ERROR GOTO 0
```

Se avviene un errore nel momento in cui il programma apre il file, il controllo del programma salta alla routine che si trova nell'etichetta *noQueFile:*

```
noQueFile:
   noTrov% = TRUE
   PRINT
   PRINT "Non si trova questo file nel disco."
RESUME NEXT
```

Questa routine assegna un valore logico vero alla variabile logica *noTrov%* e visualizza sullo schermo un semplice messaggio d'errore. Poi l'istruzione RESUME NEXT passa il controllo alla linea immediatamente dopo l'istruzione che attivò la gestione degli errori. (Per ulteriori dettagli consultare la voce RESUME.) Nel programma principale, questa istruzione stabilisce se il programma può riuscire a leggere il file:

```
IF noTrov% THEN END
```

In altre parole, se la gestione degli errori è stata attivata e *noTrov%* contiene un valore logico vero, il programma è terminato. Altrimenti, se *noTrov%* è falsa, l'esecuzione continua ed il programma conta il numero di linee nel file aperto. Un altro programma che illustra la gestione degli errori è listato nella voce della funzione DATE (Capitolo 24, Parte VII). Il programma riceve le stringhe della data e dell'orario immesse dall'utente alla tastiera e poi usa le istruzioni DATE$ e TIME$ per cambiare le impostazioni attuali per la data e l'orario del sistema. Poiché entrambe le istruzioni DATE$ e TIME$ danno

errori run-time se si lavora con stringhe non valide, il programma usa due diverse gestioni degli errori per ognuna di queste due istruzioni. Ad esempio, ecco il frammento che dà una nuova stringa della data:

```
DO
   dataOK% = TRUE
   LINE INPUT "Immettere una nuova data del sistema: ", inData$
   ON ERROR GOTO DataErr
      IF LTRIM$(inData$) <> "" THEN DATE$ = inData$
   ON ERROR GOTO 0
LOOP UNTIL dataOK%
```

In questo caso la routine *DataErr* assegna semplicemente un valore logico falso alla variabile logica *dataOK%*:

```
DataErr:
   dataOK% = FALSE
RESUME NEXT
```

Il ciclo LOOP nel programma principale visualizza sullo schermo il messaggio che richiede l'input ripetutamente fin che *dataOK%* è vera. Una gestione degli errori simile è attivata per il processo di cambiamento dell'impostazione dell'orario.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione ON ERROR. In BASICA la routine di gestione degli errori può essere identificata solo da un numero della linea, non da un'etichetta.

RESUME

L'istruzione RESUME termina l'attività di una routine di gestione degli errori e specifica la posizione in cui il controllo verrà ritornato.

```
RESUME 0        ' Ritorna il controllo all'istruzione che
                ' ha causato l'errore. 0 è opzionale.

RESUME NEXT     ' Passa il controllo all'istruzione dopo
                ' quella che ha causato l'errore.
```

```
RESUME num     ' Passa il controllo ad un numero di linea specificato.

RESUME etic    ' Passa il controllo ad un'etichetta di linea specificata.
```

Una routine di gestione degli errori è attivata da un'istruzione ON ERROR; la routine viene eseguita solo se avviene un errore run-time mentre la gestione degli errori è attiva. Alla fine di questa routine, un'istruzione RESUME indica in che punto il programma dovrebbe continuare. RESUME da sola (o con l'argomento 0) passa il controllo all'istruzione che in origine attivò la gestione degli errori. RESUME NEXT passa il controllo all'istruzione seguente. Queste sono le due sintassi dell'istruzione RESUME più comunemente usate in QuickBASIC. (Inoltre, un'istruzione RESUME può passare il controllo ad un numero o ad un'etichetta di linea specificati, ma questa tecnica può dare una struttura confusa del programma.)

Un esempio di RESUME

Questa routine di gestione degli errori semplicemente stabilisce il valore di una variabile logica e poi ritorna il controllo all'istruzione che si trova dopo quella che causò l'errore:

```
DataErr:
   dataOK% = FALSE
RESUME NEXT
```

(Vedere la voce ON ERROR per ulteriori approfondimenti di questo esempio.)

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione RESUME.

STOP

L'istruzione STOP termina l'esecuzione del programma. STOP viene usato talvolta in una routine di gestione degli errori se un programma non riesce a riprendere da un errore che è stato commesso.

```
STOP
```

In alcuni contesti, STOP è semplicemente un'alternativa al comando END. Si può usare anche STOP come uno strumento diagnostico in ambiente QuickBASIC. Dopo che un'istruzione STOP ha terminato l'esecuzione del programma, si può premere il tasto funzione F5 per continuare l'esecuzione. (Diversamente da END, l'istruzione STOP non chiude i file aperti.) Si noti che il menù di Debug di QuickBASIC offre strumenti che sono molto più utili e sofisticati di STOP.

Compatibilità

Sia Turbo Basic sia BASICA hanno il comando STOP. In BASICA, STOP visualizza il numero della linea in cui il programma è terminato.

TRON E TROFF

TRON e TROFF sono strumenti diagnostici che attivano e disattivano la stampa dei numeri di linea delle istruzioni durante l'esecuzione. Queste istruzioni sono essenzialmente fuori moda rispetto agli strumenti sofisticati ora disponibili nel programma di Debug incorporato in QuickBASIC.

```
TRON
TROFF
```

Il comando TRON fa sì che QuickBASIC evidenzi il numero di linee di ogni istruzione durante l'esecuzione del programma. L'istruzione TROFF ripristina il funzionamento normale. Il comando Trace On nel menù Debug è un commutatore che sostituisce TRON e TROFF. Quando il comando è attivato, ogni istruzione viene evidenziata quando è eseguita. Se Trace è disattivato, l'esecuzione avviene normalmente.

Compatibilità

Sia Turbo Basic sia BASICA hanno le istruzioni TRON e TROFF.

Capitolo 23

Gestione degli eventi

La gestione degli eventi è una delle tante tecniche di QuickBASIC progettate per gestire gli eventi esterni. Uno dei tipi più comuni di gestione degli eventi risponde a speciali sequenze di tasti che l'utente può premere da tastiera. La gestione degli eventi dalla tastiera è oggetto della prima parte di questo capitolo, mentre tutti gli altri tipi di eventi vengono descritti nella seconda parte.

GESTIONE DEGLI EVENTI GENERATI DA TASTIERA

La gestione degli eventi generati da tastiera è una valida alternativa ad altre tecniche d'input più convenzionali. Questo tipo di gestione degli eventi è adatto soprattutto quando il programma ha bisogno di rispondere a sequenze di controllo non ASCII della tastiera. Per stabilire una gestione degli eventi da tastiera si usano tre diverse istruzioni QuickBASIC, di solito in questo ordine:

1. L'istruzione KEY definisce la sequenza di tasti, scelta dall'utente, che alla fine farà partire la gestione.
2. L'istruzione ON KEY GOSUB crea la gestione ed identifica il sottoprogramma che il programma chiamerà, se l'utente preme la sequenza di tasti definita.

3. L'istruzione KEY(n) ON attiva la gestione e l'istruzione KEY(n) OFF la disattiva.

Le voci seguenti descrivono tutte queste istruzioni e forniscono esempi di tecniche di gestione degli eventi da tastiera.

KEY

L'istruzione KEY ha quattro diversi utilizzi (uno solo dei quali è direttamente collegato alle tecniche di gestione degli eventi generati da tastiera). Ognuno di essi ha la propria speciale sintassi: (1) La sintassi KEY *n, str* indica un valore *tasti dedicati* per uno dei dodici tasti funzione. (2) KEY ON visualizza nella linea inferiore dello schermo l'attuale attribuzione ai primi dieci tasti funzione dedicati. KEY OFF disattiva ancora questo video. (3) KEY LIST dà una visualizzazione verticale di tutti i dodici tasti funzione dedicati. (4) La sintassi KEY *n, tasti* crea una sequenza di tasti definita dall'utente per utilizzarla in una successiva gestione degli eventi.

(1) Attribuzione dei valori ai tasti dedicati:

```
KEY n, str  ' Assegna un valore stringa al tasto funzione numero n.
```

(2) Visualizzazione orizzontale dell'attribuzione dei valori ai tasti funzione:

```
KEY ON      ' Attiva la visualizzazione orizzontale del tasto
            ' funzione nella linea inferiore dello schermo.

KEY OFF     ' Disattiva la visualizzazione orizzontale del tasto
            ' funzione nella linea inferiore dello schermo.
```

(3) Visualizzazione verticale dell'attribuzione dei valori ai tasti funzione:

```
KEY LIST    ' Dà una lista verticale di tutti i dodici tasti funzione.
```

(4) Crea una sequenza da tastiera definita dall'utente:

```
KEY n, tasti   ' Assegna un numero di tasto, n, associato
               ' alle combinazioni di tasti specificati.
```

Nella prima parte vengono trattati i tre utilizzi dell'istruzione KEY, nella seconda l'uso di KEY nel contesto della gestione degli eventi generati da tastiera.

Attribuzione e visualizzazione dei valori dei tasti dedicati

QuickBASIC non ha un'attribuzione predefinita per i dieci tasti funzione (contrassegnati da F1 a F10) sulla tastiera originale PC IBM o per i due tasti funzione supplementari (F11 e F12) sulla tastiera estesa. Si possono usare i vari formati dell'istruzione KEY per stabilire l'attribuzione per questi tasti e per renderli disponibili all'utente in un programma particolare. Per convenienza dell'utente, si può anche visualizzare sullo schermo questa attribuzione. Ecco la sintassi di KEY per assegnare valori stringa ai tasti funzione:

```
KEY n, str
```

Il valore di *n* è un intero che va da 1 a 10 e che rappresenta uno dei tasti funzione da F1 a F10. Per le tastiere estese, gli interi 30 e 31 rappresentano rispettivamente F11 e F12. L'argomento *str*, che può contenere fino a quindici caratteri, diventa il valore d'input che il programma riceve quando l'utente schiaccia il tasto funzione F*n*. Se si desidera che venga eseguito un ritorno automatico del carrello alla fine della pressione del tasto dedicato, bisogna specificarlo nell'istruzione KEY; ad esempio:

```
KEY n, strTast$ + CHR$(13)
```

Il numero di codice ASCII del carattere del ritorno del carrello è 13. Quando questo carattere viene aggiunto alla fine del valore del tasto dedicato, l'utente non deve premere il tasto Invio per completare l'input dal tasto funzione. I comandi KEY ON e KEY LIST sono due diversi modi di visualizzare sullo schermo i valori tasto dedicato una volta definiti. KEY ON posiziona una visualizzazione orizzontale dei primi dieci tasti funzione nella linea inferiore dello schermo. (KEY OFF disattiva ancora questa visualizzazione.) KEY LIST, d'altra parte, visualizza una lista verticale delle attribuzioni di tutti i dodici tasti funzione.

Alcuni esempi di KEY, KEY ON e KEY LIST

Questo esercizio illustra una tecnica per attribuire i valori ai tasti dedicati e per visualizzarli sullo schermo:

```
CONST INVIO = 13

READ max%
FOR numTast% = 1 TO max%
   READ attrTast%
   KEY numTast%, attrTast$ + CHR$(INVIO)
NEXT numTast%

CLS
KEY LIST
PRINT
PRINT USING "Premere un tasto funzione da F1 a F##"; max%
INPUT "per selezionare un'operazione sul database. -> ", op$
END

DATA 10
DATA Apri, Mostra, Modifica, Aggiungi, Cancella
DATA Comprimi, Chiudi, Crea_Indice, Ordina, Esci
```

Il ciclo FOR nella parte superiore di questo programma legge una sequenza di valori stringa dei tasti dedicati dalle linee DATA. L'istruzione KEY stabilisce ogni stringa come il valore per un dato tasto funzione, da F1 a F10. (Si noti che il programma concatena un carattere di ritorno del carrello alla fine di ogni valore dei tasti dedicati.) Un'istruzione KEY LIST visualizza sullo schermo i valori dei tasti dedicati. Infine, un'istruzione INPUT chiede all'utente di premere uno di questi tasti; il valore d'input risultante viene memorizzato nella variabile stringa *op$*. Ecco un esempio d'attivazione del programma:

```
F1 Apri
F2 Mostra
F3 Modifica
F4 Aggiungi
F5 Cancella
F6 Comprimi
F7 Chiudi
F8 Crea_Indice
F9 Ordina
F10 Esci
F11
F12
```

```
Premere un tasto funzione da F1 a F10 per selezionare
un'operazione sul database. -> Cancella
```

Per ripristinare tutti i tasti funzione al loro stato originale bisogna assegnare una stringa vuota ad ogni tasto dedicato. La seguente routine ha questa funzione:

```
FOR i% = 1 TO 10
   KEY i%, ""
NEXT i%
KEY 30, ""
KEY 31, ""
```

Come definire i tasti che attivano la gestione degli eventi

Il quarto formato dell'istruzione KEY crea una sequenza da tastiera che attiverà la successiva routine di gestione degli eventi:

```
KEY n, tasti
```

Qui *n* è un intero che va da 15 a 25 e rappresenta uno dei tasti definibili dall'utente. Il secondo argomento, *tasti* è un valore stringa di due byte, in cui entrambi i byte vengono di solito espressi come chiamate alla funzione CHR$:

```
CHR$(maius) + CHR$(tast)
```

Il primo byte chiamato *flag della tastiera* rappresenta un tasto delle maiuscole come Shift, oppure Ctrl o Alt (tasti modificatori).

```
CHR$(0)     ' no maiuscolo

CHR$(1)     ' Tasti maiuscole
CHR$(2)
CHR$(3)
CHR$(4)     ' il tasto Ctrl
CHR$(8)     ' il tasto Alt
CHR$(32)    ' il tasto Bloc Num
CHR$(64)    ' il tasto Caps Lock
```

Si possono aggiungere più codici per indicare delle combinazioni di tasti modificatori. Ad esempio, CHR$(12) rappresenta Ctrl-Alt.

Il secondo byte è il numero del *codice della tastiera* per un tasto che sarà premuto contemporaneamente al tasto modificatore. Il codice di tastiera del DOS, da non confondere con il codice ASCII, rappresenta con un intero ogni singolo tasto della tastiera. Tutti i codici di tastiera sono elencati nell'Appendice B. Ecco alcuni esempi importanti per il frammento di programma presentato in seguito:

```
CHR$(28)  ' Il tasto Invio.
CHR$(72)  ' Il tasto freccia verso l'alto.
CHR$(75)  ' Il tasto freccia verso sinistra.
CHR$(77)  ' Il tasto freccia verso destra.
CHR$(80)  ' Il tasto freccia verso il basso.
```

Il compito specifico dell'istruzione KEY in questo formato è di assegnare un numero di tasto, *n*, ad una combinazione di tasti che attiverà una gestione degli eventi della tastiera. Il valore *n* comparirà di conseguenza come argomento dell'istruzione ON KEY(n) che crea la gestione degli eventi ed anche nella corrispondente istruzione KEY(n) ON che attiva la gestione.

Alcuni esempi di KEY

Il programma *GestioneDate* (listato nel Capitolo 31) crea una gestione degli eventi per una specifica serie di combinazioni di tasti, molte delle quali comprendono il tasto Ctrl. Queste speciali combinazioni di tasti rappresentano le istruzioni specifiche che l'utente può dare al programma. Ad esempio, l'utente può premere il tasto Ctrl insieme al tasto freccia verso destra per dire al programma di visualizzare sullo schermo il *mese* successivo del calendario o il tasto Ctrl con il tasto freccia verso sinistra per vedere il mese *precedente*. Allo stesso modo, Ctrl-(freccia verso l'alto) è un'istruzione per visualizzare il mese corrente dell'anno precedente e Ctrl-(freccia verso il basso) richiede il mese corrente dell'anno successivo. Il primo passo per stabilire queste gestioni degli eventi è quello di assegnare i numeri dei tasti definibili dall'utente (che, in questo esempio, vanno da 15 a 24) alle combinazioni stesse dei tasti. Ecco il frammento preso da *GestioneDate* che esegue questo compito:

```
noMaius$ = CHR$(0)   ' Valore nullo.
ctrl$ = CHR$(4)      ' Tasto-Ctrl.
```

```
KEY 15, ctrl$ + CHR$(72)     ' Ctrl-SU
KEY 16, ctrl$ + CHR$(75)     ' Ctrl-SINISTRA
KEY 17, ctrl$ + CHR$(77)     ' Ctrl-DESTRA
KEY 18, ctrl$ + CHR$(80)     ' Ctrl-GIU'
KEY 19, noMaius$ + CHR$(72)  ' SU
KEY 20, noMaius$ + CHR$(75)  ' SINISTRA
KEY 21, noMaius$ + CHR$(77)  ' DESTRA
KEY 22, noMaius$ + CHR$(80)  ' GIU'
KEY 23, noMaius$ + CHR$(28)  ' INVIO
KEY 24, ctrl$ + CHR$(28)     ' Ctrl-INVIO
```

Una volta stabilite queste definizioni dei tasti, il programma usa le istruzioni ON KEY(n) e KEY(n) ON per attivare ogni gestione degli eventi. Ad esempio, ecco le istruzioni che creano la gestione per l'istruzione Ctrl-SU:

```
ON KEY(15) GOSUB AnnoPrec    ' Ctrl-SU
KEY(15) ON
```

Vedere le voci ON KEY(n) e KEY(n) ON per ulteriori informazioni riguardo a questi esempi.

Compatibilità

Sia Turbo Basic sia BASICA supportano tutti i formati dell'istruzione KEY. In BASICA ci sono solo sei numeri di tasti definibili dall'utente, che vanno da 15 a 20.

ON KEY(N) GOSUB

Il comando ON KEY(n) GOSUB crea una gestione degli eventi generati da tastiera ed identifica la corrispondente routine di gestione degli eventi.

```
ON KEY(n) GOSUB num     ' num è la posizione della routine
                        ' di gestione degli eventi.

ON KEY(n) GOSUB etic    ' etic è la posizione della routine
                        ' di gestione degli eventi.
```

ON KEY(n) GOSUB indica la posizione di un sottoprogramma che il programma chiamerà dopo che l'utente premerà uno specifico tasto o una

combinazione di tasti. Si può identificare il sottoprogramma con un numero o un'etichetta di linea. (Le etichette rappresentano lo stile preferito di QuickBASIC.) Ogni istruzione ON KEY(n) è generalmente seguita da una corrispondente istruzione KEY(n) ON che attiva la gestione degli eventi. (Vedere le voci KEY(n) ON/OFF/STOP per ulteriori dettagli.) L'argomento *n* è, sia in ON KEY(n) GOSUB sia in KEY(n) ON, un intero compreso nei seguenti intervalli:

da 1 a 10	Per i tasti funzione F1 e F10
30 e 31	Per i tasti funzione F11 e F12
da 11 a 14	Per i tasti freccia del cursore
da 15 a 25	Per le combinazioni a tastiera definibili dall'utente

Si usa l'istruzione KEY per creare i tasti definibili dall'utente.

Un esempio di ON KEY(n) GOSUB

Il programma *GestioneDate* (listato nel Capitolo 31) crea la gestione degli eventi per molte diverse combinazioni di tasti, che danno all'utente una serie di semplici e sicure tecniche per fornire istruzioni al programma. Per ogni gestione, il programma prima usa un'istruzione KEY per definire la combinazione dei tasti, poi un'istruzione ON KEY(n) GOSUB per definire la gestione ed infine un'istruzione KEY(n) ON per attivarla. Ad esempio, questa istruzione KEY assegna il numero 15, alla combinazione Ctrl-SU:

```
ctrl$ = CHR$(4)
KEY 15, ctrl$ + CHR$(72)
```

Un'istruzione ON KEY(n) seguente specifica che il programma chiamerà il sottoprogramma che si trova nell'etichetta *AnnoPrec* quando l'utente preme questi tasti:

```
ON KEY(15) GOSUB AnnoPrec
```

Una corrispondente istruzione KEY(n) ON attiva la gestione:

```
KEY(15) ON
```

In effetti, queste istruzioni servono a QuickBASIC per interrompere l'esecuzione del programma quando l'utente preme Ctrl-(tasto freccia verso l'alto) (cioè i due tasti contemporaneamente) e per chiamare il sottoprogramma localizzato nell'etichetta *AnnoPrec*. Questo sottoprogramma trasla la videata del calendario indietro di dodici mesi:

```
AnnoPrec:
   IF annoCorr% > MINYEAR% THEN
      annoCorr% = annoCorr% - 1
      giorMax% = giorMese%(meseCorr%, annoCorr%)
      IF giorCorr% > giorMax% THEN giorCorr% = giorMax%
      VisualMese meseCorr%, giorCorr%, annoCorr%
   END IF
RETURN
```

L'istruzione RETURN, alla fine di questo sottoprogramma, riporta alla posizione del programma che aveva il controllo prima che *AnnoPrec* fosse chiamato.

Compatibilità

Sia Turbo Basic sia BASICA hanno il comando ON KEY(n) GOSUB.

KEY(N) ON/OFF/STOP

L'istruzione KEY(n) è progettata per attivare, disattivare o posporre l'attività di gestione degli eventi stabilita da un corrispondente comando ON KEY(n) GOSUB.

```
KEY(n) ON      ' Attiva la gestione dei tasti.
KEY(n) OFF     ' Disattiva la gestione dei tasti.
KEY(n) STOP    ' Pospone la gestione dei tasti.
```

Le istruzioni KEY(n) ON, KEY(n) OFF e KEY(n) STOP possono comparire dopo l'istruzione ON KEY(n) GOSUB che crea una gestione degli eventi. Un'istruzione KEY(n) ON attiva la gestione e fa sì che il programma inizi il monitoraggio per specifiche battute di tasti. Se l'utente preme i tasti in questione, il programma chiama il sottoprogramma di gestione degli eventi

identificato nel comando ON KEY(n) GOSUB. Un'istruzione KEY(n) OFF disattiva la gestione, così il programma non risponderà più alle battute di tasti. L'istruzione KEY(n) STOP blocca la gestione degli eventi, ma solo temporaneamente. Il programma tiene in considerazione ogni battuta di tasti significativa eseguita dall'utente dopo un comando KEY(n) STOP. Quando una successiva istruzione KEY(n) ON riattiva la gestione degli eventi, il programma risponde alle battute di tasti premuti mentre la gestione era stata bloccata. In questo senso, KEY(n) STOP è un modo di posporre l'attività di gestione dei tasti finché il programma non è pronto. L'argomento *n* è, sia nell'istruzione KEY(n) sia in KEY(n) ON GOSUB, un intero compreso nei seguenti intervalli:

da 1 a 10	Per i tasti funzione F1 e F10
30 e 31	Per i tasti funzione F11 e F12
da 11 a 14	Per i tasti freccia
da 15 a 25	Per le combinazioni a tastiera definibili dall'utente

L'istruzione KEY stabilisce le combinazioni dei tasti definibili dall'utente.

Alcuni esempi di KEY(n) ON/OFF/STOP

Il programma *GestioneDate* (listato nel Capitolo 31) crea la gestione degli eventi per molte diverse combinazioni di tasti, che danno all'utente una serie di semplici e sicure tecniche a tastiera per fornire istruzioni al programma. Per ogni gestione, il programma prima usa un'istruzione KEY per definire la combinazione stessa dei tasti, poi un'istruzione ON KEY(n) GOSUB per determinare la gestione ed infine un'istruzione KEY(n) ON per attivarla. Ad esempio, questa istruzione KEY definisce la gestione per la combinazione Ctrl-(freccia verso l'alto):

```
ctrl$ = CHR$(4)
KEY 15, ctrl$ + CHR$(72)

ON KEY(15) GOSUB AnnoPrec
KEY (15) ON
```

(Vedere la voce ON KEY(n) GOSUB per approfondire il sottoprogramma di gestione degli eventi e la voce KEY per informazioni sui tasti definiti dall'utente.)

Compatibilità

Sia Turbo Basic sia BASICA hanno il comando KEY(n) ON.

ALTRE GESTIONI DEGLI EVENTI

Oltre alle tecniche di gestione degli eventi a tastiera, QuickBASIC fornisce strumenti per gestire queste operazioni:

- Input da un dispositivo di comunicazioni seriali.
- Input da una penna ottica.
- Input da un joystick.
- Modifica nello stato del buffer di memoria di PLAY.
- Il trascorrere del tempo, come misurato dall'orologio del sistema.
- Un evento definito dall'utente.

Le tecniche di gestione degli eventi per queste operazioni hanno tutte una struttura simile. Le due voci finali del capitolo riassumono queste tecniche.

ON EVENTO GOSUB

Ogni comando ON *evento* GOSUB crea una gestione per un particolare tipo di evento ed identifica il corrispondente sottoprogramma di gestione degli eventi.

```
ON COM(n) GOSUB num    ' Gestione degli eventi per comunicazioni seriali.
ON COM(n) GOSUB etic

ON PEN GOSUB num       ' Gestione degli eventi per una penna ottica.
ON PEN GOSUB etic
```

```
ON PLAY(n) GOSUB num     ' Gestione degli eventi per l'istruzione PLAY.
ON PLAY(n) GOSUB etic

ON STRIG(n) GOSUB num    ' Gestione degli eventi per un joystick.
ON STRIG(n) GOSUB etic

ON TIMER(n) GOSUB num    ' Gestione degli eventi per TIMER.
ON TIMER(n) GOSUB etic

ON UEVENT GOSUB num      ' Gestione degli eventi per un
                         ' evento definito dall'utente.
ON UEVENT GOSUB etic
```

Ognuna di queste istruzioni identifica un sottoprogramma che il programma chiamerà ogni volta che avviene un'evento specificato. La posizione del sottoprogramma può essere espressa come numero o etichetta di una linea. (QuickBASIC preferisce le etichette.) Ogni istruzione ON *evento* di solito è seguita da una corrispondente istruzione *evento* ON che attiva la gestione dell'evento. (Vedere la voce *evento* ON/OFF/STOP per ulteriori dettagli.) L'istruzione ON COM(n) GOSUB definisce una gestione degli eventi per operazioni di comunicazione su una porta seriale specificata, dove *n* può essere 1 o 2. Quando questa gestione è attiva (COM(n) ON), il programma salta al sottoprogramma specificato ogni volta che la porta riceve i dati. (Vedere anche la voce OPEN COM nel Capitolo 13.) L'istruzione ON PEN GOSUB definisce una gestione per input da una penna ottica. Quando questa gestione è attiva (PEN ON), il programma salta al sottoprogramma specificato, in risposta alle operazioni della penna ottica eseguite dall'utente. (La funzione PEN dà informazioni sullo stato del commutatore e la posizione delle coordinate della penna ottica. Per ulteriori dettagli vedere la parte riguardante la funzione PEN nel Capitolo 13.) L'istruzione ON PLAY(n) GOSUB definisce una gestione per controllare il comando PLAY, ma solo quando sta producendo della musica di sottofondo. Se la gestione è attiva (PLAY ON), il programma salta al sottoprogramma specificato quando per PLAY sono rimaste in memoria meno di *n* note musicali da eseguire. Il valore di *n* è un intero da 1 a 32. (Per ulteriori dettagli consultare la voce PLAY nel Capitolo 14. QuickBASIC ha anche una funzione PLAY, che indica il numero di note disponibili da suonare.) L'istruzione ON STRIN(n) GOSUB definisce una gestione per input da un joystick. In questa istruzione il valore *n* identifica un pulsante d'attivazione alzato o abbassato su uno dei due joystick: i valori *n* 0 e 4 rappresentano i pulsanti sul primo joystick, mentre i valori 2 e 6 quelli sul secondo. Quando questa gestione è attiva, il

programma salta al sottoprogramma specificato ogni volta che l'utente preme il pulsante indicato. Un programma che lavora con i joystick dovrebbe attivare contemporaneamente le gestioni per tutti e quattro i pulsanti. (La funzione STRIG legge le informazioni da uno specificato pulsante del joystick, sebbene questa funzione *non* dovrebbe essere usata come parte di una routine di gestione degli eventi. La funzione STICK fornisce la posizione delle coordinate dei joystick. Per ulteriori dettagli vedere le voci STRIG e STICK nel Capitolo 13.) L'istruzione ON TIMER(n) GOSUB definisce una gestione che risponde al trascorrere del tempo, come misurato dall'orologio del sistema. Qui il valore di *n* rappresenta i secondi, da 1 a 86.400, il numero di secondi in un giorno. Se questa gestione è attiva (TIMER ON), il programma salta al sottoprogramma specificato quando *n* secondi sono trascorsi dall'ultima chiamata. L'istruzione ON UEVENT GOSUB stabilisce una gestione per un evento definito dall'utente. L'evento stesso viene definito da una serie di routine esterne in linguaggio assembler, spesso per controllare le attività di un particolare dispositivo hardware. Il comando UEVENT ON attiva la gestione.

Un esempio ON *evento* GOSUB

Il programma *GestioneDate* (presentato nel Capitolo 31) contiene una gestione degli eventi, il cui compito è visualizzare sullo schermo l'orario corrente in un formato digitale che cambia ogni secondo. Queste istruzioni creano ed attivano la gestione:

```
ON TIMER(1) GOSUB VisualOra
TIMER ON
```

In effetti, queste istruzioni fanno sì che QuickBASIC interrompa l'esecuzione del programma ogni secondo e chiami il sottoprogramma che si trova sull'etichetta *VisualOra*. Questo sottoprogramma stabilisce di visualizzare l'ora corrente in un punto fissato sul video e in un formato a.m./p.m.:

```
VisualOra:
   VIEW PRINT
   LOCATE 23, 12

'  Converte l'orario in 24 ore nel formato a.m./p.m.
```

```
   oraCorr% = VAL(LEFT$(TIME4, 2))
   minSec$ = RIGHT$(TIME$, 6)
   SELECT CASE oraCorr%
      CASE 0
         PRINT "12"; minSec$; " a.m."
      CASE 1 TO 11
         PRINT USING "##& a.m."; oraCorr%; minSec$
      CASE 12
         PRINT TIME$ + " p.m."
      CASE ELSE
         PRINT USING "##& p.m."; oraCorr% - 12; minSec$
   END SELECT
RETURN
```

L'istruzione RETURN alla fine di questo sottoprogramma ritorna alla posizione del programma che era in controllo prima che fosse chiamata *VisualOra*.

Compatibilità

Sia Turbo Basic sia BASICA hanno i comandi ON *evento* GOSUB, ad eccezione di ON UEVENT, che è nuovo in QuickBASIC 4.5.

EVENTO ON/OFF/STOP

Ogni istruzione *evento* è progettata per attivare, disattivare o posporre le operazioni di gestione degli eventi stabilite da un corrispondente comando ON *evento* GOSUB.

```
COM(n) ON      ' Attiva le comunicazioni dalla porta n.
COM(n) OFF     ' Disattiva le comunicazioni.
COM(n) STOP    ' Pospone le comunicazioni.

PEN ON         ' Attiva la gestione degli eventi relativa
               ' alla penna ottica.
PEN OFF        ' Disattiva la gestione.
PEN STOP       ' Pospone la gestione.

PLAY ON        ' Attiva la gestione degli eventi relativa a PLAY.
PLAY OFF       ' Disattiva la gestione.
PLAY STOP      ' Pospone la gestione.
STRIG(n) ON    ' Attiva la gestione degli eventi relativa al joystick.
```

```
STRIG(n) OFF  ' Disattiva la gestione.
STRIG(n) STOP ' Pospone la gestione.

TIMER ON      ' Attiva la gestione degli eventi relativa al TIMER.
TIMER OFF     ' Disattiva la gestione.
TIMER STOP    ' Pospone la gestione.

UEVENT ON     ' Attiva la gestione di un evento definito dall'utente.
UEVENT OFF    ' Disattiva la gestione.
UEVENT STOP   ' Pospone la gestione.
```

Dopo l'istruzione ON *evento* GOSUB, che crea la gestione di un evento, può comparire qualsiasi numero di istruzioni *evento* ON, *evento* OFF e *evento* STOP. Un'istruzione *evento* ON attiva la gestione e mette il programma in allarme per l'evento specificato. Se l'evento accade, il programma chiama il sottoprogramma di gestione degli eventi identificato nel comando ON *evento* GOSUB. Un'istruzione *evento* OFF disattiva la gestione, in modo che il programma non risponda più all'evento specificato. L'istruzione *evento* STOP blocca anche la gestione, ma solo temporaneamente. Il programma prende in considerazione ogni evento importante che avviene dopo un comando *evento* STOP. Quando una conseguente istruzione *evento* ON riattiva la gestione, il programma risponde ad ogni evento che si verifica mentre la gestione era interrotta. In questo senso, *evento* STOP è un modo di posporre l'operazione di gestione degli eventi finché il programma non è pronto. Per ulteriori dettagli sui tipi specifici di gestioni degli eventi disponibili in QuickBASIC, consultare la voce ON *evento* GOSUB. (Inoltre, vedere la voce ON KEY(n) GOSUB.)

Alcuni esempi di evento ON/OFF/STOP

Il programma *GestioneDate* (presentato nel Capitolo 31) stabilisce una gestione TIMER, responsabile della visualizzazione dell'ora sulla parte inferiore dello schermo. Il programma cambia l'ora ogni secondo, a meno che un'operazione rapida da tastiera non prenda temporaneamente la precedenza sulla gestione TIMER. Queste istruzioni stabiliscono la gestione:

```
ON TIMER(1) GOSUB VisualOra
TIMER ON
```

(Vedere ON *evento* GOSUB per ulteriori approfondimenti sul sottoprogramma di gestione degli eventi.) La procedura in *GestioneDate* che visualizza sullo schermo il calendario del mese in corso deve temporaneamente disattivare la gestione TIMER per evitare un'interruzione indesiderata nelle operazioni video della procedura stessa.
Questa procedura si chiama *VisualMese* e la sua prima istruzione eseguibile è:

```
TIMER OFF
```

Quando la procedura completa il suo output allo schermo, l'operazione di gestione TIMER viene ripristinata:

```
TIMER ON
```

Compatibilità

Sia Turbo Basic sia BASICA hanno le istruzioni *evento* GOSUB, ad eccezione di UEVENT ON/OFF/STOP, che sono nuove in QuickBASIC 4.5.

Parte VII

Controllo del sistema operativo

I quattro capitoli che compongono la Parte VII contengono una completa esposizione degli strumenti di controllo del sistema operativo di cui dispone QuickBASIC. Inoltre, vengono trattati alcuni degli aspetti più tecnici ed avanzati del linguaggio.

- Il Capitolo 24 passa in rassegna le funzioni e le istruzioni che consentono di accedere all'ora ed alla data di sistema.
- Il Capitolo 25 spiega le istruzioni e le funzioni che consentono di eseguire le principali funzioni di sistema del DOS dall'interno di un programma QuickBASIC. Questi strumenti consentono di eseguire specifici compiti riguardanti le unità a disco, le directory ed i file.
- Il Capitolo 26 presenta le procedure di basso livello di QuickBASIC che consentono di accedere direttamente alle locazioni della memoria centrale consentendo ai programmi di leggere e modificare il codice e i dati presenti in memoria.
- Infine, il Capitolo 27 descrive le istruzioni che eseguono chiamate a routine scritte in linguaggi diversi da QuickBASIC, e le istruzioni che eseguono le chiamate di sistema DOS.

La Parte VII è una breve rassegna di questi strumenti e non una introduzione completa delle tecniche di programmazione che li utilizzano. Per ulteriori informazioni, il lettore può consultare i manuali tecnici del DOS ed i libri che discutono la programmazione in linguaggi diversi da QuickBASIC.

Capitolo 24

Operazioni con data e ora

L'attribuzione di valori alla data corrente e all'ora su computer è generalmente controllata a livello del sistema operativo. Comunque, il linguaggio QuickBASIC ha una serie limitata di funzioni ed istruzioni che consentono un utile accesso a queste attribuzioni. In particolare, le funzioni DATE$, TIME$ e TIMER permettono ad un programma QuickBASIC di modificare la data e l'orario attuale del sistema e le istruzioni DATE$ e TIME$ danno al programma la possibilità di cambiare i valori di data ed ora. Questo capitolo descrive dettagliatamente questi cinque strumenti e presenta degli esempi presi dai programmi principali di questo libro. Il Capitolo 24 prende in considerazione solo gli strumenti incorporati di QuickBASIC che danno accesso alla data e dall'orario del sistema. Per quanto riguarda gli algoritmi relativi, occorre consultare altri capitoli del libro. Ad esempio, QuickBASIC *non* fornisce routine generali di data ed orario per eseguire importanti operazioni, come quelle aritmetiche sulle date e relativi confronti, nonché le conversioni del formato per la visualizzazione della data e dell'orario. Si possono, però, sviluppare tali routine da soli, come illustrato nei programmi *Compleanno*, *Mese*, *Agenda*, *GestioneDate*, quattro dei principali esempi di programma presentati nella Parte VIII.

LA FUNZIONE DATE$

La funzione DATE$ ritorna un valore stringa che rappresenta la data corrente del sistema.

```
DATE$
```

La funzione non ha argomenti e ritorna una stringa di dieci caratteri, di cui i primi due sono le cifre che rappresentano il mese, il quarto ed il quinto il giorno e gli ultimi quattro l'anno. I caratteri in terza e sesta posizione sono lineette che separano i vari elementi della data. Si possono usare le funzioni LEFT$, MID$ e RIGHT$ per estrarre i tre elementi della stringa DATE$ e la funzione VAL per convertire questi elementi in interi che rappresentano il mese, il giorno e l'anno. Ad esempio, si consideri questo frammento per il programma *Agenda* (presentato nel Capitolo 30):

```
mese% = VAL(LEFT$(DATE$, 2))
giorno% = VAL(MID$(DATE$, 4, 2))
anno% = VAL(RIGHT$(DATE$, 4))
```

Queste istruzioni danno tre valori interi che rappresentano la data corrente. Come illustrato nelle applicazioni, ad esempio il programma *Agenda*, separare questi tre interi è un primo passo fondamentale per lavorare con le date.

Un esempio della funzione DATE$

Il seguente programma, chiamato *DataOrario*, visualizza sullo schermo la data e l'orario correnti e dà all'utente l'opportunità di cambiarli. Il programma illustra le funzioni DATE$ e TIME$ e le istruzioni DATE$ e TIME$:

```
' DATAORARIO.BAS
' Esegue un dialogo d'input per cambiare data e ora del sistema.

CONST FALSE = 0: TRUE = NOT FALSE

CLS
PRINT STRING$(32, "*")
PRINT " Cambiare le attribuzioni correnti"
PRINT " per la data e l'orario del sistema."
PRINT STRING$(32, "*")
PRINT

PRINT "Attribuzione della data corrente: "; DATE$

DO
   dataOK% = TRUE
   LINE INPUT "Immettere la nuova data del sistema: ", inData$
```

```
   ON ERROR GOTO DataErr
      IF LTRIM$(inData$) <> "" THEN DATE$ = inData$
   ON ERROR GOTO 0
LOOP UNTIL dataOK%
PRINT STRING$(32, "-")
PRINT "Valore della nuova data: "; DATE$

PRINT
PRINT STRING$(32, "=")
PRINT

PRINT "Valore dell'orario corrente: "; TIME$

DO
   oraOK% = TRUE
   LINE INPUT "Immettere il nuovo orario di sistema: ", inOra$
   ON ERROR GOTO OraErr
      IF LTRIM$(inOra$) <> "" THEN TIME$ = inOra$
   ON ERROR GOTO 0
LOOP UNTIL oraOK%
PRINT STRING$(32, "-")
PRINT "Valore del nuovo orario: "; TIME$

END

DataErr:
   dataOK% = FALSE
RESUME NEXT

OraErr:
   oraOK% = FALSE
RESUME NEXT
```

Il programma visualizza sullo schermo la data corrente per due volte, la prima volta prima che l'utente immetta una nuova data:

```
PRINT "Valore della data corrente: "; DATE$
```

e poi dopo che la data è stata cambiata:

```
PRINT "Valore della nuova data: "; DATE$
```

Ecco l'esecuzione della prima parte del programma:

```
**********************************
Cambiare le attribuzioni correnti
per la data e l'orario di sistema.
**********************************
```

```
Attribuzione della data corrente: 01-01-1980
Immettere la nuova data del sistema: 7/1/89
-------------------------------------------
Valore della nuova data: 07-01-1989
```

Vedere l'istruzione DATE$ e le due voci TIME$ per ulteriori dettagli riguardo a questo programma.

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione DATE$.

L'ISTRUZIONE DATE$

L'istruzione DATE$ cambia l'attribuzione corrente della data del sistema.

```
DATE$ = str
```

La stringa della nuova data si trova a destra rispetto al segno di uguale nell'istruzione DATE$. Questa stringa può comparire come un valore literal, una variabile o un'espressione. In generale, la stringa data deve seguire lo stesso formato della stringa ritornata dalla funzione DATE$, tranne per il fatto che si possono usare come delimitatori la lineetta o lo slash e l'anno può essere espresso con due o quattro cifre. Un messaggio d'errore run-time ("Chiamata di funzione errata") appare sullo schermo, se si fornisce una stringa della data non valida. L'intervallo possibile delle date del DOS va da 1-1-1980 a 31-12-2099.

Un esempio di istruzione DATE$

Il programma *DataOrario* (presentato nel paragrafo della funzione DATE$) dà un esempio dell'istruzione DATE$. Dall'interno di un ciclo DO, il programma riceve una stringa utente da tastiera e cerca di farla diventare la nuova data del sistema:

```
DO
   dataOK% = TRUE
```

```
   LINE INPUT "Immettere la nuova data del sistema: ", inData$
   ON ERROR GOTO DataErr
      IF LTRIM$(inData$) <> "" THEN DATE$ = inData$
   ON ERROR GOTO 0
LOOP UNTIL dataOK%
```

Viene attivata una gestione degli errori quando l'utente immette una data errata. Questa gestione attribuisce la variabile logica *dataOK%* a FALSE e poi torna il controllo al programma principale:

```
OraErr:
   oraOK% = FALSE
RESUME NEXT
```

Il ciclo continua e la richiesta d'input viene visualizzata ripetutamente sullo schermo finché l'iterazione viene eseguita senza attivare la gestione degli errori, cioè finché l'utente non immette una nuova data valida. Si noti, comunque, che l'utente può semplicemente premere il tasto Invio in risposta a richiesta d'input per indicare che la data non deve essere cambiata.

```
IF LTRIM$(inData$) <> "" THEN DATE$ = inData$
```

Se l'input è una stringa vuota o una stringa che contiene solo spazi vuoti, il programma non cerca di ripristinare la data.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione DATE$.

LA FUNZIONE TIME$

La funzione TIME$ ritorna un valore stringa che rappresenta l'orario di sistema corrente.

```
TIME$
```

La funzione non ha argomenti e ritorna una stringa a otto caratteri che indica l'orario nel formato a 24 ore. I primi due caratteri in questo valore di ritorno rappresentano l'ora, il quarto e il quinto i minuti e gli ultimi due i secondi.

Il carattere in terza e sesta posizione è i due punti (:) che separa gli elementi della stringa orario.

Un esempio di TIME$

Il programma *DataOrario* presentato nel paragrafo funzione DATE$ visualizza sullo schermo la data e l'orario correnti e dà all'utente l'opportunità di cambiarli. Il programma visualizza l'orario per due volte, la prima volta prima che l'utente immetta una nuova ora:

```
PRINT "Valore dell'orario corrente: "; TIME$
```

e poi dopo che l'orario è stato cambiato:

```
PRINT "Valore del nuovo orario: "; TIME$
```

Queste visualizzazioni si hanno nella seconda metà di un'attivazione, ad esempio:

```
Valore dell'orario corrente: 00:00:23
Immettere un nuovo orario di sistema: 14:24
*******************************************
Valore del nuovo orario: 14:24:00
```

Un altro esempio interessante della funzione TIME$ appare nel programma *GestioneDate*, presentato nel Capitolo 32. Il programma stabilisce una gestione degli errori che passa il controllo ad un sottoprogramma chiamato *VisualOra* dopo ogni secondo trascorso:

```
ON TIMER(1) GOSUB VisualOra
TIMER ON
```

Il sottoprogramma *VisualOra* visualizza l'orario corrente in un punto stabilito dello schermo ed organizza la conversione del formato standard a 24 ore in un formato a.m./p.m.:

```
VisualOra:
   VIEW PRINT
   LOCATE 23, 12

' Convertire l'orario in 24 ore nel formato a.m./p.m.
```

```
    oraCorr% = VAL(LEFT$(TIME$, 2))
    minSec$ = RIGHT$(TIME$, 6)
    SELECT CASE oraCorr%
       CASE 0
          PRINT "12"; minSec$; " a.m."
       CASE 1 TO 11
          PRINT USING "##& a.m."; oraCorr%; minSec$
       CASE 12
          PRINT TIME$ + " p.m."
       CASE ELSE
          PRINT USING "##& p.m."; oraCorr% - 12; minSec$
    END SELECT
RETURN
```

L'effetto risultante è una visualizzazione digitale dell'orario che cambia ogni secondo ed è in grado di distinguere correttamente tra a.m. e p.m. I seguenti esempi indicano la differenza tra la visualizzazione data da *GestioneDate* e il valore di ritorno equivalente della funzione TIME$.

Visualizzazione GestioneDate	*Equivalente TIME$*
12:19:25 a.m.	00:19:25
6:45:09 a.m.	06:45:09
12:52:35 p.m.	12:52:35
3:19:22 p.m.	15:19:22
11:05:58 p.m.	23:05:59

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione TIME$.

L'ISTRUZIONE TIME$

L'istruzione TIME$ cambia l'attribuzione corrente dell'orario di sistema.

```
TIME$ = str
```

La stringa del nuovo orario si trova a destra rispetto al segno di uguale nell'istruzione TIME$. Questo valore può comparire come una stringa literal, una variabile o un'espressione; deve corrispondere al formato a 24 ore dato dalla funzione TIME$, tranne per il fatto che si può omettere la seconda parte della stringa. Un messaggio d'errore run-time ("Chiamata di funzione errata") appare sullo schermo, se si fornisce una stringa dell'orario non valida.

Un esempio di TIME$

Il programma *DataOrario* (presentato nel paragrafo della funzione DATE$) dà un esempio dell'istruzione TIME$. Dall'interno di un ciclo DO, il programma riceve una stringa dell'orario dall'utente a tastiera e cerca di farla diventare il nuovo orario di sistema:

```
DO
   oraOK% = TRUE
   LINE INPUT "Immettere il nuovo orario del sistema: ", inOra$
   ON ERROR GOTO OraErr
      IF LTRIM$(inOra$) <> "" THEN TIME$ = inOra$
   ON ERROR GOTO 0
LOOP UNTIL oraOK%
```

Viene attivata una gestione degli errori quando l'utente immette un orario errato. Questa gestione semplicemente attribuisce la variabile logica *oraOK%* a FALSE e poi passa il controllo al programma principale:

```
OraErr:
   oraOK% = FALSE
RESUME NEXT
```

Il ciclo continua finché l'iterazione viene eseguita senza attivare la gestione degli errori, cioè finché l'utente non immette un nuovo orario valido. Si noti, comunque, che l'utente può semplicemente premere il tasto Invio in risposta al prompt d'input per indicare che l'orario non dovrebbe essere cambiato.

```
IF LTRIM$(inOra$) <> "" THEN TIME$ = inOra$
```

Se l'input è una stringa vuota o una stringa che contiene solo spazi vuoti, il programma non cerca di modificare l'orario.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione TIME$.

TIMER

La funzione TIMER ritorna un valore intero da 0 a 86.400, che rappresenta il numero di secondi trascorsi dalla mezzanotte del giorno corrente.

```
TIMER
```

La funzione non ha argomenti e il risultato è un valore intero lungo. TIMER è usato nei programmi che hanno bisogno di sapere quanto tempo ci vuole per eseguire specifici compiti.

Un esempio di TIMER

Questo esercizio dà una dimostrazione dell'utilizzo di TIMER nelle istruzioni PRINT e nei calcoli aritmetici. Il programma visualizza il tempo trascorso dalla mezzanotte considerando i secondi, i minuti e le ore.

```
CLS
DO
   LOCATE 8, 22
   PRINT "L'orario corrente è "; TIME$; "."
   LOCATE 9, 10
   PRINT USING "##.### secondi "; TIMER;
   PRINT USING "(o #.### minuti pieni, o "; TIMER \ 60;
   PRINT USING "## ore piene)"; TIMER \ 3.600
   LOCATE 10, 17
   PRINT "sono trascorsi dalla mezzanotte di "; DATE$; "."
LOOP UNTIL INKEY$ <> ""
```

La visualizzazione del programma cambia ogni secondo e rimane sullo schermo finché non si preme un tasto. Ecco un esempio di output:

```
                    L'orario corrente è 17:36:03.
         63.363 secondi (o 1.056 minuti pieni o 17 ore piene)
           sono trascorsi dalla mezzanotte del 07-11-1990.
```

Compatibilità

Sia Turbo Basic sia BASICA (versione 2.0 e successive) supportano la versione TIMER.

Capitolo 25

Strumenti d'interazione con il DOS

Questo capitolo esamina le istruzioni e le funzioni QuickBASIC che sono equivalenti a specifici comandi del DOS. Questi strumenti consentono di eseguire operazioni essenziali del DOS dall'interno di un programma QuickBASIC:

- L'istruzione FILES visualizza sullo schermo una lista di nomi di file da un indirizzario specificato. Nel DOS lo strumento parallelo a questa istruzione è il comando DIR.
- L'istruzione NAME cambia il nome di un file esistente. La stessa operazione viene eseguita dal comando RENAME nel DOS.
- L'istruzione KILL elimina uno o più file dal disco. Nel DOS si usa il comando ERASE per cancellare i file.
- Le istruzioni CHDIR, MKDIR e RMDIR sono progettate per cambiare o togliere un indirizzario dal disco. Gli equivalenti comandi del DOS sono CD, MD e RD.
- L'istruzione ENVIRON e la funzione ENVIRON$ consentono l'accesso alle stringhe dell'ambiente DOS. Insieme questi due strumenti sono equivalenti al comando SET del DOS.

Inoltre, questo capitolo descrive tre istruzioni che passano il controllo dal programma corrente ad un altro del DOS o di QuickBASIC. Il comando SHELL, forse dei tre il più utile e il più comunemente usato, esegue un

comando esterno del DOS e poi ritorna il controllo al programma corrente di QuickBASIC. Le altre due istruzioni sono CHAIN e RUN: entrambe forniscono tecniche per saltare dal programma in corso ad un altro. Nelle singole voci di questo capitolo vi sono descrizioni ed esempi di tutti gli strumenti correlati al DOS.

CHAIN

L'istruzione CHAIN trasferisce il controllo ad un altro programma QuickBASIC e facoltativamente passa i dati al nuovo programma usando le variabili COMMON.

```
CHAIN progr    ' progr è un valore stringa che assegna
               ' il nome del programma, con un nome
               ' opzionale per l'indirizzario.
```

Il parametro *progr* può presentarsi come un valore stringa literal, una variabile o un'espressione stringa. L'operazione risultante dipende dal fatto che il programma sia eseguito all'interno dell'ambiente QuickBASIC o dal prompt del DOS:

- Nell'ambiente di sviluppo, QuickBASIC cerca sul disco il file del codice sorgente, con estensione BAS per default. Se lo trova, il programma viene caricato nell'editor di QuickBASIC (sostituendo il programma che contiene l'istruzione CHAIN) ed eseguito.
- Dal prompt del DOS, il programma cerca il file programma eseguibile con un'estensione EXE per default. Se lo trova, il programma viene eseguito.

Si noti che, dopo l'esecuzione di un programma concatenato, CHAIN non ritorna il controllo al programma originale. CHAIN consente al primo programma di trasferire i dati al secondo attraverso le variabili COMMON. Perché questa operazione abbia successo, entrambi i programmi devono avere le opportune istruzioni COMMON con liste di variabili. (Vedere la voce COMMON per ulteriori informazioni.)

Un esempio di CHAIN

Il seguente programma è progettato come dimostrazione dell'istruzione CHAIN. Il programma inizia conducendo un breve dialogo d'input e poi concatena uno degli altri tre programmi, a seconda della scelta dell'utente:

```
' Dimostrazione dell'istruzione CHAIN.

COMMON nomeSoc$

CLS
LINE INPUT "Qual è il nome della società? "; nomeSoc$

PRINT
PRINT "Quale database si vuole esaminare?"
PRINT
PRINT "  I)mpiegati"
PRINT "  C)lienti"
PRINT "  V)enditori"
PRINT
PRINT "       ?";
LOCATE , , 1
DO
   db$ = UCASE$(INKEY$)
LOOP UNTIL db$ <> "" AND INSTR("ECV", db$) <> 0

SELECT CASE db$
   CASE "I"
      CHAIN "ImpDB"
   CASE "C"
      CHAIN "CliDB"
   CASE "V"
      CHAIN "VendDB"
END SELECT
```

Se questo programma viene compilato ed attivato dal prompt del DOS, concatena IMPDB.EXE, CLIDB.EXE o VENDDB.EXE. Ognuno di questi programmi dovrebbe avere un'istruzione COMMON per ricevere il valore stringa che questo programma memorizza nella variabile stringa *nomeSoc$*. Il nome della variabile deve essere lo stesso nel programma concatenato; ad esempio:

```
'IMPDB.EXE
COMMON soc$
```

Quando il programma concatena IMPDB.EXE, il valore memorizzato in *nomeSoc$* verrà passato alla variabile stringa *soc$*.

Compatibilità

L'istruzione CHAIN di Turbo Basic lavora solo con i programmi compilati e non può essere utilizzata all'interno di un ambiente di sviluppo di Turbo Basic. L'istruzione CHAIN in BASICA concatena sempre un file codice sorgente con un'estensione BAS per default. La versione BASICA di quest'istruzione ha molte opzioni che sono disponibili anche in QuickBASIC. Ad esempio, un programma BASICA si può concatenare ad un numero della linea specificato di un altro programma e passare tutte le variabili definite all'altro programma.

CHDIR

L'istruzione CHDIR cambia l'indirizzario di default su un particolare drive.

```
CHDIR indir    ' indir è una stringa con meno di 128 caratteri.
```

Si può esprimere l'argomento *indir* come un valore stringa literal, una variabile o una concatenazione. Se si include una specificazione del drive nella stringa *indir*, CHDIR cambia l'indirizzario di default sul drive. Comunque, lo stato attuale del drive rimane invariato.

Un esempio di CHDIR

Il seguente programma è progettato come dimostrazione dell'utilizzo di CHDIR e di molti altri strumenti correlati al DOS. Il programma crea un indirizzario temporaneo chiamato \CHRON sul drive D: cambia l'indirizzario corrente su D: e crea due file nell'indirizzario. Poi, dopo alcune altre ipotetiche operazioni, il programma sposta i file da D:\CHRON a D:\ ed elimina l'indirizzario D:\CHRON. Durante il funzionamento del programma vengono visualizzati alcuni messaggi in vari punti dello schermo. Ecco il listato del programma:

```
' Una dimostrazione delle seguenti istruzioni:
' CHDIR, RMDIR, MKDIR, FILES, NAME, KILL.

' Crea un indirizzario temporaneo.

MKDIR "D:\CHRON"
CHDIR "D:\CHRON" ' Notare: non cambia il drive corrente.

' Crea un file temporaneo.

OPEN "D:TEMP1.DAT" FOR OUTPUT AS #1
OPEN "D:TEMP2.DAT" FOR OUTPUT AS #2
WRITE #1, DATE$
WRITE #2, TIME$
CLOSE #1, #2

' ... altre operazioni che interessano questo
' indirizzario temporaneo.

' Visualizzare l'indirizzario.

CLS
PRINT "-> Ecco l'indirizzario di D:\CHRON"
PRINT " dopo la creazione di due file:"
PRINT
FILES "D:"

' Sposta il file, ne cambia i nomi e visualizza i due indirizzari.

NAME "D:TEMP1.DAT" AS "D:\DATA.DAT"
NAME "D:TEMP2.DAT" AS "D:\ORA.DAT"
PRINT
PRINT "-> Ecco lo stesso indirizzario dopo"
PRINT " che i file sono stati spostati:"
PRINT
FILES "D:"

PRINT
PRINT "-> Ecco l'indirizzario D:\*.DAT dopo"
PRINT " che i file sono stati spostati e ridenominati:"
PRINT
FILES "D:\*.DAT"

' Elimina l'indirizzario.

CHDIR "D:\"
RMDIR "D:\CHRON"

' Cancella i file da D:\

KILL "D:DATA.DAT"
KILL "D:ORA.DAT"
```

Il programma cambia per due volte l'indirizzario di default sul drive D:. La prima modifica avviene subito dopo che il nuovo indirizzario è stato creato:

```
MKDIR "D:\CHRON"
CHDIR "D:\CHRON"
```

Dopo questo cambiamento, ogni specificazione del drive D: si riferisce a D:\CHRON per default. Ad esempio, ecco come il programma apre i due file in questo nuovo indirizzario:

```
OPEN "D:TEMP1.DAT" FOR OUTPUT AS #1
OPEN "D:TEMP2.DAT" FOR OUTPUT AS #2
```

La seconda modifica dell'indirizzario avviene appena prima che il programma sia pronto ad eliminare l'indirizzario \CHRON dal drive D:

```
CHDIR "D:\"
RMDIR "D:\CHRON"
```

Questa variazione è necessaria, perché RMDIR non può eliminare un indirizzario che è attualmente quello di default. (Vedere le voci RMDIR e MKDIR per ulteriori informazioni.) Questi sono i messaggi che il programma visualizza sullo schermo durante le sue operazioni:

```
-> Ecco l'indirizzario di D:\CHRON
   dopo la creazione di due file:

D:\CHRON
   .  <DIR> .. <DIR> TEMP1 .DAT TEMP2 .DAT

31.719.424 Byte liberi

-> Ecco lo stesso indirizzario dopo
   che i file sono stati spostati:

D:\CHRON
   . <DIR> .. <DIR>
31.719.424 Byte liberi

-> Ecco l'indirizzario D:\*.DAT dopo
   che i file sono stati spostati e ridenominati:

D:\CHRON
DATE .DAT TIME .DAT
31.719.424 Byte liberi
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione CHDIR.

ENVIRON$ E ENVIRON

La funzione ENVIRON$ ritorna una stringa dalla tabella d'ambiente del DOS. L'istruzione ENVIRON assegna un nuovo parametro alla tabella d'ambiente.

```
ENVIRON$(str)  ' Ritorna il valore della stringa ambiente.

ENVIRON$(num)  ' Ritorna il valore della stringa
               ' ambiente specificata dal numero.

ENVIRON str    ' Aggiunge una stringa all'ambiente.
```

L'argomento della funzione ENVIRON$ può essere sia una stringa sia un intero. L'argomento stringa è il nome di un'esistente stringa ambiente del DOS. Un argomento intero è il numero (iniziando da 1) di uno dei parametri d'ambiente. In ogni caso, ENVIRON$ ritorna il valore corrente della stringa d'ambiente specificata. L'istruzione ENVIRON cambia il valore di un parametro d'ambiente. L'argomento stringa di questa istruzione si presenta in una di queste forme:

```
"nome=attribuz"
"nome attribuz"
```

dove *nome* è il nome della stringa d'ambiente e *attribuz* è il nuovo valore assegnato.

Alcuni esempi di ENVIRON

Questo programma utilizza l'istruzione ENVIRON per cambiare l'attribuzione della stringa PATH e poi usa la funzione ENVIRON$ per visualizzare tutti i valori correnti delle stringhe ambiente:

```
ENVIRON "PATH=C:\DOS;C\QB45"
```

```
CLS
i% = 1
DO
   PRINT ENVIRON$(i%)
   i% = i% + 1
LOOP UNTIL ENVIRON$(i%) = ""
```

Ecco un esempio dell'output che questo programma visualizza sullo schermo:

```
COMSPEC=C:\DOS\COMMAND.COM
APPEND=C:\DOS
PATH=C:\DOS;C\QB45
```

Compatibilità

Turbo Basic e GW-BASIC hanno l'istruzione ENVIRON e la funzione ENVIRON$. Le prime versioni di BASICA non le avevano.

FILES

L'istruzione FILES visualizza una lista dei nomi dei file di un particolare indirizzario.

```
FILES          ' Visualizza tutti i file dall'indirizzario di default.

FILES descr    ' descr è una stringa che può contenere una
               ' specificazione del drive, il nome di
               ' un indirizzario e dei caratteri jolly.
```

La stringa opzionale *descr* stabilisce il contenuto della lista risultante dei file. Si possono usare i caratteri jolly * e ? per specificare quali file si desidera elencare. L'asterisco rappresenta un numero qualunque di caratteri nel nome di un file ed ogni punto interrogativo un singolo carattere. Nella stringa *descr* si possono includere anche il nome del drive e dell'indirizzario. Se si omette questa stringa, FILES visualizza tutti i file nell'indirizzario di default.

Un esempio di FILES

La voce CHDIR contiene un programma che descrive l'istruzione FILES. Il programma crea un nuovo indirizzario chiamato \CHRON sul drive D: e memorizza nell'indirizzario due nuovi file. Un'istruzione CHDIR cambia l'indirizzario di default in D: con \CHRON. Poi visualizza tutti i file al momento memorizzati nell'indirizzario:

```
FILES "D:"
```

In seguito il programma utilizza questa istruzione per visualizzare tutti i file con estensione DAT nell'indirizzario radice di D:

```
FILES "D:\*.DAT"
```

Vedere la voce CHDIR per esaminare l'output che si ha da queste istruzioni.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione FILES.

KILL

L'istruzione KILL elimina i file da un indirizzario e da un drive specificati.

```
KILL descr      ' descr è una stringa che può contenere
                ' una specificazione del drive, il nome di
                ' un indirizzario e dei caratteri jolly.
```

La stringa *descr* specifica quale o quali file verranno eliminati. Si possono usare i caratteri jolly * e ? per identificare un gruppo di file con nomi simili. L'asterisco rappresenta un numero qualunque di caratteri nel nome di un file ed ogni punto interrogativo un singolo carattere. KILL elimina tutti i file che corrispondono alle specificazioni date con i caratteri jolly. Nella stringa *descr* si possono includere anche il nome del drive e dell'indirizzario.

Un esempio di KILL

La voce CHDIR contiene un programma che descrive l'istruzione KILL. Il programma crea alcuni file temporanei chiamati DATE.DAT e TIME.DAT sul drive D: e cancella i file dal disco prima di terminare:

```
KILL "D:DATE.DAT"
KILL "D:TIME.DAT"
```

Se questi due file sono i soli ad avere estensione DAT, questa istruzione darebbe risultati uguali:

```
KILL "D:*.DAT"
```

Comunque, nell'istruzione KILL si dovrebbero usare i caratteri jolly solo quando si è sicuri che non si perderà involontariamente nessun file.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione KILL.

MKDIR

L'istruzione MKDIR crea un nuovo indirizzario su un particolare drive.

```
MKDIR indir      ' indir è una stringa con meno di 128 caratteri.
```

Si può esprimere l'argomento *indir* come un valore stringa literal, una variabile o una concatenazione. Se si include una specificazione del drive nella stringa *indir*, MKDIR non cambia l'indirizzario di default sul drive.

Un esempio di MKDIR

Un programma d'esempio nella voce CHDIR contiene una dimostrazione dell'istruzione MKDIR. Il programma crea un nuovo indirizzario chiamato

\CHRON sul drive D: e poi cambia l'indirizzario di default in D: con \CHRON:

```
MKDIR "D:\CHRON"
CHDIR "D:\CHRON"
```

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione MKDIR.

NAME

L'istruzione NAME assegna un nuovo nome ad un file esistente o sposta un file in un nuovo indirizzario sullo stesso drive.

```
NAME vec AS nuo   ' vec è il nome attuale del file e
                  ' nuo è quello nuovo.
```

Si possono esprimere gli argomenti *vec* e *nuo* come valori stringa literal, variabili o espressioni. La stringa *vec* deve essere il nome del file, che al momento deve rimanere chiuso. La stringa *nuo* deve essere un nome che attualmente non appartiene a nessun altro file dello stesso indirizzario. Sia *vec* sia *nuo* possono contenere specificazioni d'indirizzario. Se i due indirizzari non sono uguali, il file viene spostato in quello specificato nella stringa *nuo*. (Dopo questa operazione si troverà il file nel nuovo indirizzario con il nuovo nome e non esisterà più nell'indirizzario vecchio.) Per spostare un file da un drive all'altro non si può usare NAME.

Un esempio di NAME

Un programma listato nella voce CHDIR contiene una dimostrazione dell'istruzione NAME. Queste istruzioni spostano i file chiamati TEMP1.DAT e TEMP2.DAT dall'indirizzario di default del drive D: all'indirizzario radice del drive:

```
NAME "D:TEMP1:DAT" AS "D:\DATE.DAT"
NAME "D:TEMP2:DAT" AS "D:\TIME.DAT"
```

Dopo queste due istruzioni, i nuovi nomi dei file sono DATE.DAT e TIME.DAT.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione NAME.

RMDIR

L'istruzione RMDIR elimina un indirizzario da un particolare drive.

```
RMDIR indir   ' indir è una stringa con meno di 128 caratteri.
```

Si può esprimere l'argomento *indir* come un valore stringa literal, una variabile o una concatenazione. Se si include una specificazione del drive nella stringa *indir*, RMDIR elimina l'indirizzario da quel drive. L'indirizzario in questione deve essere vuoto e non deve essere l'indirizzario di default sul drive specificato.

Un esempio di RMDIR

Il programma d'esempio nella voce CHDIR contiene una dimostrazione dell'istruzione RMDIR. Il programma crea un indirizzario temporaneo chiamato \CHRON sul drive D:. Dopo alcune operazioni sui file in \CHRON, il programma elimina tutti i file dall'indirizzario ed infine l'indirizzario stesso dal drive:

```
CHDIR "D:\"
RMDIR "D:\CHRON"
```

Si noti che il programma si sposta nell'indirizzario radice nel drive D: prima di cercar di eliminare l'indirizzario \CHRON. Questo passo è necessario perché l'indirizzario corrente non può essere eliminato.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione RMDIR.

RUN

L'istruzione RUN passa il controllo ad un altro programma o ad un'istruzione specificata nel programma corrente.

```
RUN prog     ' Passa il controllo al programma identificato
             ' dall'argomento stringa prog.

RUN num      ' Fa ripartire il programma corrente dalla linea num.

RUN          ' Fa ripartire il programma corrente iniziando
             ' dal modulo principale del programma.
```

L'argomento *prog* può comparire come un valore stringa literal, una variabile o un'espressione stringa. Contiene il nome del programma in questione oltre al nome (opzionale) dell'indirizzario in cui si trova. Il funzionamento risultante dipende dal fatto che si stia lavorando all'interno dell'ambiente di sviluppo QuickBASIC o nel prompt del DOS:

- Nell'ambiente di sviluppo, QuickBASIC cerca sul disco un file sorgente, con estensione BAS per default. Se lo trova, il programma viene caricato nell'editor di QuickBASIC (sostituendo il programma che contiene l'istruzione RUN) ed eseguito. In questo caso, il codice sorgente deve essere un programma QuickBASIC.
- Dal prompt del DOS, il programma cerca un file del programma compilato con un'estensione EXE per default. Se lo trova il programma viene eseguito. In questo caso, il file può essere qualsiasi programma compilato e non deve corrispondere ad una sorgente QuickBASIC.

Dopo che l'esecuzione del secondo programma è completa, RUN non ritorna il controllo al programma originale. In alternativa, l'argomento dell'istruzione RUN può essere un numero di linea a cui verrà trasferito il controllo nel programma corrente. (Non sono ammesse le etichette.) RUN fà ripartire il programma da un punto specificato. L'istruzione RUN chiude

i file aperti e cancella tutte le variabili prima di attivare (o riattivare) il programma in questione. RUN non consente di passare i dati tra due programmi. A parte questo, RUN è simile all'istruzione CHAIN.

Un esempio di RUN

Il seguente programma è progettato come dimostrazione dell'istruzione RUN. Il programma inizia conducendo un breve dialogo d'input e poi si concatena ad uno degli altri tre programmi, a seconda della scelta dell'utente:

```
' Dimostrazione dell'istruzione RUN.

CLS
PRINT "Società XYZ"
PRINT "-----"
PRINT

PRINT "Quale database si desidera esaminare?"
PRINT
PRINT "  I)mpiegati"
PRINT "  C)lienti"
PRINT "  V)enditori"
PRINT
PRINT "       ?";
LOCATE , , 1
DO
   db$ = UCASE$(INKEY$)
LOOP UNTIL db$ <> "" AND INSTR("ECV", db$) <> 0

SELECT CASE db$
   CASE "I"
      RUN "ImpDB"
   CASE "C"
      RUN "CliDB"
   CASE "V"
      RUN "VendDB"
END SELECT
```

Se questo programma viene compilato ed attivato dal prompt del DOS, passa il controllo a IMPDB.EXE, CLIDB.EXE o VENDDB.EXE.

Compatibilità

L'istruzione RUN di Turbo Basic lavora solo con i programmi compilati trasferendo il controllo tra due diversi programmi. L'istruzione RUN in BASICA trasferisce il controllo ad un file del codice sorgente con un'estensione BAS per default o ad un numero di linea nel programma corrente.

SHELL

L'istruzione SHELL provoca un'uscita temporanea al DOS dal programma corrente.

```
SHELL          ' Passa il controllo al prompt del DOS.

SHELL com      ' Esegue un singolo comando del DOS e poi
               ' ritorna al programma corrente.
```

L'argomento opzionale *com* è un valore stringa che contiene il nome di un comando (interno o esterno) del DOS. SHELL può eseguire un programma che ha un'estensione COM o EXE o un file batch con estensione BAT. Inoltre, la stringa *com* può essere il nome di un comando interno del DOS come DIR o COPY. Se si omette l'argomento *com*, SHELL semplicemente va temporaneamente al DOS. Si può eseguire qualsiasi operazione dal prompt del DOS. Poi con il comando EXIT si ritorna al programma:

```
C>EXIT
```

Il programma QuickBASIC riprende l'esecuzione dall'istruzione immediatamente dopo il comando SHELL.

Un esempio di SHELL

Il programma *GestioneDate*, listato ed esaminato nel Capitolo 32, usa l'istruzione SHELL per passare il controllo al programma *Agenda*:

```
SHELL "AGENDA" + Comlin$
```

GestioneDate prepara una stringa di istruzioni da inviare ad *Agenda*, questa stringa viene memorizzata nella variabile *Comlin$*. *Agenda*, invece, legge queste istruzioni attraverso una chiamata alla funzione COMMAND$. Per funzionare con successo, il programma *Agenda* deve esistere su disco in forma compilata sotto il nome di AGENDA.EXE. (Vedere la Parte VIII per ulteriori dettagli.)

Compatibilità

Turbo Basic e GW-BASIC hanno l'istruzione SHELL, mentre BASICA no.

SYSTEM

L'istruzione SYSTEM termina l'esecuzione di un programma.

```
SYSTEM
```

L'effetto di SYSTEM è uguale a quello di END in QuickBASIC. Ogni file aperto viene chiuso, il programma è finito e si ritorna all'ambiente di sviluppo QuickBASIC. In un file compilato EXE, SYSTEM passa il controllo al DOS.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione SYSTEM. In BASICA, SYSTEM esegue un'uscita dall'editor e dall'ambiente di programmazione ritornando al DOS.

Capitolo 26

Gestione della memoria

Questo capitolo descrive un gruppo di istruzioni e funzioni QuickBASIC che eseguono alcune operazioni di basso livello per la gestione della memoria. Questi strumenti sono importanti nelle applicazioni avanzate che hanno a che fare direttamente con i valori memorizzati in specifici indirizzi della memoria o in programmi che legano porzioni scritte in QuickBASIC con il codice creato da altri linguaggi. Molte di queste procedure fanno direttamente riferimento alle locazioni della memoria ad accesso casuale. Un indirizzo nella memoria è costituito da due parti: l'indirizzo iniziale di un *segmento* e uno *spiazzamento* in quel segmento. Un segmento è un blocco contiguo di memoria con una lunghezza massima di 64Kb. L'istruzione DEF SEG indica il segmento corrente in un programma QuickBASIC. La porzione di spiazzamento di un indirizzo della memoria è un intero da 0 a 65.535 e rappresenta un indirizzo nell'attuale segmento. Ad esempio, i riferimenti alla memoria in PEEK, POKE, BLOAD e BSAVE sono tutti spiazzamenti del segmento indicato.

BSAVE E BLOAD

L'istruzione BSAVE salva i contenuti di un blocco contiguo di byte di memoria su disco (o su alcuni altri dispositivi) come un file "d'immagine di memoria". BLOAD legge questi file e ne memorizza il contenuto in una specifica locazione della memoria.

```
BSAVE file, ind, byte   ' file è un valore stringa; ind rappresenta
                        ' uno spiazzamento nel segmento corrente e
                        ' byte il blocco di memoria.

BLOAD file, ind         ' L'opzionale parametro ind è lo spiazzamento
                        ' con cui verrà memorizzato il contenuto
                        ' del file.
```

Il nome del file, in entrambe queste istruzioni, è espresso come un valore stringa literal, come una variabile stringa o un'espressione. (Si può usare l'estensione che si preferisce.) Il valore *ind* è uno spiazzamento nel segmento di memoria corrente, come definito dall'istruzione DEF SEG. L'istruzione BSAVE salva il contenuto di un blocco di memoria, di lunghezza pari a *byte*, dove *byte* è un valore da 0 a 65.535. BSAVE può creare un nuovo file o sovrascriverne uno esistente, se il file esiste già. L'istruzione BLOAD legge un file che contiene un'immagine di memoria, creato da BSAVE. Il file viene caricato in memoria a partire da uno specifico indirizzo, sovrascrivendo le informazioni già memorizzate a partire da quell'indirizzo. Se si omette l'indirizzo dopo BLOAD, il file viene memorizzato allo stesso indirizzo da cui era stato letto in origine. (L'istruzione BSAVE registra automaticamente l'indirizzo originale in ogni file d'immagine della memoria.) Le istruzioni BSAVE e BLOAD vengono comunemente usate insieme alle istruzioni (grafiche) GET e PUT, per creare e leggere i file d'immagini grafiche. Qui di seguito vi è un esempio.

Alcuni esempi BSAVE e BLOAD

La voce per le istruzioni (grafiche) GET e PUT contiene un programma d'esempio che usa l'istruzione PUT per visualizzare un'immagine grafica sullo schermo. Questi due programmi, che danno una dimostrazione dell'utilizzo di BSAVE e BLOAD, sono adattati da quell'esempio. Il primo programma disegna un paio di immagini in SCREEN 9 ed usa GET per salvare entrambe le immagini in un solo vettore. Poi il programma, con l'istruzione BSAVE, memorizza il contenuto del vettore su disco in un file chiamato FACC.MEM:

```
' Dimostrazione di BSAVE.

DECLARE SUB Facc (att%)
```

```
CONST sorr = 0
CONST tris = 2
CONST modSch% = 9

SCREEN modSch%
COLOR 1, 15

DIM immFac(1828)
' DISEGNA la faccia sorridente e poi SALVALA CON GET.

Faccia sorridente
GET (324, 164)-(406, 246), immFac
CLS

' DISEGNA la faccia triste e poi SALVALA CON GET.

Faccia triste
GET (324, 164)-(406, 246), immFac(915)

SCREEN 0

' Salva l'immagine grafica sul disco nel file FACC.MEM.

DEF SEG = VARSEG(immFacc(0))
BSAVE "facc.mem", VARPTR(immFacc(0)), 7312

END

' Queste linee DATA contengono istruzioni DRAW per la
' faccia sorridente e per quella triste.

DATA 4
DATA F1 D1 F1 D1 F1 D1 F1 D1 F1 D1 F5
DATA F1 R1 F1 R1 F1 R1 F1 R1 F1 R1 R10
DATA E1 R1 E1 R1 E1 R1 E1 R1 E1 R1 E5
DATA E1 U1 E1 U1 E1 U1 E1 U1 E1 U1

SUB Facc (att%)

' Il sottoprogramma Facc disegna sullo schermo una faccia
' triste o sorridente.

' Legge le istruzioni DRAW dalle linee DATA.

RESTORE
disLin$ = ""
READ linInf%
FOR i% = 1 TO linInf%
   READ d$
   disLin$ = disLin$ + d$ + " "
NEXT i%
```

```
' Disegna la faccia.

PSET (339, 204)
IF att% = tris% THEN DRAW "BM+50,+15"
DRAW "A=" + VARPTR$(att%) + " C4"
DRAW disLin$

CIRCLE (355, 190), 3, 4
CIRCLE (375, 190), 3, 4
CIRCLE (365, 205), 40, 4
PAINT (365, 190), 15, 4

END SUB
```

Il programma inizia disegnando le immagini grafiche ed usando il comando GET per salvare le immagini in un vettore numerico a precisione semplice chiamato *immFac*. Per salvare il contenuto di questo vettore sul disco, il programma inizia designando l'indirizzo del segmento del vettore come segmento corrente:

```
DEF SEG = VARSEG(immFac(0))
```

In questa istruzione, la funzione VARSEG ritorna il segmento in cui è memorizzato il vettore. L'istruzione DEF SEG stabilisce poi il segmento corrente. Si noti che l'elemento iniziale del vettore viene esplicitamente identificato tramite *immFac(0)*. Il programma poi usa la funzione VARPTR per precisare l'indirizzo (inteso come spiazzamento nel segmento corrente) del vettore nell'istruzione BSAVE che salva il file dell'immagine di memoria su disco.

```
' Dimostrazione di BLOAD.

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST modSch% = 9

SCREEN modSch%
COLOR 1, 15

' Definisce il vettore delle facce a precisione semplice.

dim% = 1828
DIM facc(dim%)
```

```
' Carica le immagini nel vettore dal file FACC.MEM,
' creato con BSAVE.

DEF SEG = VARSEG(facc(0))
BLOAD "facc.mem", VARPTR(facc(0))

' Stabilisce un'organizzazione a scacchiera delle facce
' sullo schermo.

buonUm% = FALSE
FOR x% = 15 TO 525 STEP 85
   buonUm% = NOT buonUm%
   FOR y% = 0 TO 255 STEP 85
      IF buonUm% THEN
         PUT (x%, y%), facc
      ELSE
         PUT (x%, y%), facc(dim% / 2 + 1), PRESET
      END IF
      buonUm% = NOT buonUm%
   NEXT y%
NEXT x%
LOCATE 25, 28
PRINT "Premere un tasto per uscire. ";
SLEEP
SCREEN 0
```

In questo caso, le immagini grafiche catturate vengono rappresentate da un vettore a precisione semplice chiamato *facc*. Un'istruzione DIM definisce il vettore *facc* e gli assegna la lunghezza richiesta:

```
dim% = 1828
DIM facc(dim%)
```

Poi il programma usa una sequenza simile di chiamate DEF SEG, VARSEG e VARPTR per definire l'indirizzo corretto per l'operazione BLOAD:

```
DEF SEG = VARSEG(facc(0))
BLOAD "facc.mem", VARPTR(facc(0))
```

L'istruzione BLOAD legge con successo il contenuto del file FACC.MEM nel vettore *facc*. Una volta che questo compito è eseguito, il programma può usare l'istruzione PUT per visualizzare le immagini sullo schermo; ad esempio:

```
PUT (x%, y%), facc
```

Vedere le voci DEF SEG, VARSEG e VARPTR e quelle grafiche GET e PUT.

Compatibilità

Sia Turbo Basic sia BASICA hanno le istruzioni BSAVE e BLOAD.

CLEAR

L'istruzione CLEAR azzera le variabili numeriche esistenti e gli elementi dei vettori, assegna valore vuoto alle variabili e ai vettori stringa e chiude i file aperti. Inoltre, un'opzione dell'istruzione CLEAR consente di controllare le dimensioni della pila.

```
CLEAR          ' Azzera le variabili e chiude i file.

CLEAR ,, dim   ' Stabilisce, in byte, le dimensioni della pila.
```

CLEAR reinizializza tutte le variabili senza influenzare le loro dichiarazioni. Chiude anche i file aperti e azzera la pila. CLEAR non può essere usato in un sottoprogramma o in una funzione. L'opzione *dim* è un intero che specifica in byte le nuove dimensioni della pila. (Le due virgole sono indicatori per vecchie opzioni CLEAR che non sono più supportate in QuickBASIC.) I programmi che usano le procedure ricorsive talvolta richiedono che le dimensioni della pila siano aumentate.

Un esempio di CLEAR

Queste linee di programma usano la funzione FRE per determinare le dimensioni della pila. La routine poi usa CLEAR per aumentare la pila se le dimensioni sono inferiori a 2.000 byte:

```
pila% = FRE(-2)
PRINT "Dimensioni della pila: "; pila%

IF pila% < 2.000 THEN CLEAR , , 2 * FRE(-2)
PRINT "Nuove dimensioni della pila: "; FRE(-2)
```

Compatibilità

L'istruzione CLEAR di Turbo Basic non ha l'opzione *dim*, mentre in BASICA l'istruzione CLEAR ha due opzioni:

```
CLEAR , dim1, dim2
```

L'opzione *dim1* stabilisce le dimensioni del segmento di dati e *dim2* rappresenta le dimensioni della pila.

DEF SEG

L'istruzione DEF SEG definisce l'indirizzo iniziale del segmento corrente in memoria. Gli indirizzi che compaiono in molte altre istruzioni e funzioni QuickBASIC sono spiazzamenti all'interno di questo segmento.

```
DEF SEG = ind     ' Designa ind come l'inizio del segmento corrente.

DEF SEG           ' Ripristina il segmento di default,
                  ' il segmento dati di QuickBASIC.
```

Gli indirizzi in memoria sono formati da due parti: l'indirizzo iniziale del segmento corrente ed uno spiazzamento in quel segmento. Un segmento è un blocco contiguo di memoria con una lunghezza massima di 64Kb. Lo spiazzamento in un indirizzo di memoria è perciò un intero da 0 a 65.535. Molte istruzioni e funzioni prendono spiazzamenti come argomenti; tra queste PEEK e POKE, BLOAD, BSAVE e CALL ABSOLUTE. Gli spiazzamenti in queste istruzioni sono effettivamente degli indirizzi all'interno del segmento corrente, come specificato da DEF SEG. Il segmento di default è il segmento di dati QuickBASIC.

Un esempio di DEF SEG

Il programma esempio listato alle voci BLOAD e BSAVE utilizza l'istruzione DEF SEG per stabilire il segmento in cui porre un vettore. Il vettore è progettato per memorizzare un'immagine grafica che il programma successivamente carica in memoria dal file FACC.MEM:

```
DIM facc(dim%)
DEF SEG = VARSEG(facc(0))
BLOAD "facc.mem", VARPTR(facc(0))
```

Consultare le voci BLOAD, BSAVE, VARPTR e VARSEG per ulteriori informazioni riguardo questo esempio.

Compatibilità

Sia Turbo Basic sia BASICA hanno l'istruzione DEF SEG.

FRE

La funzione FRE ritorna delle informazioni sull'impiego corrente della memoria.

```
FRE(str)     ' Ritorna lo spazio di memorizzazione per le stringhe.

FRE(-1)      ' Ritorna le dimensioni della memoria
             ' utilizzabili per vettori numerici.

FRE(-2)      ' Ritorna il numero di byte dello spazio
             ' inutilizzato sulla pila.

FRE(num)     ' Ritorna le dimensioni del successivo blocco libero
             ' di memoria utilizzabile per le stringhe.
             ' num è il valore -1 o -2.
```

Secondo l'argomento numerico o stringa che si fornisce, la funzione FRE ritorna le dimensioni di un'area della memoria selezionata. Se si immette un argomento della stringa, FRE specifica anche la memoria libera per memorizzare stringhe.

Un esempio di FRE

Queste linee di programma utilizzano la funzione FRE per determinare le dimensioni della pila. La routine usa poi CLEAR per aumentare la pila se le dimensioni sono inferiori ai 2.000 byte.

```
pila% = FRE(-2)
PRINT "Dimensioni della pila: "; pila%

IF pila% < 2000 THEN CLEAR , , 2 * FRE(-2)
PRINT "Nuove dimensioni della pila: "; FRE(-2)
```

Compatibilità

Sia Turbo Basic sia BASICA hanno la funzione FRE.

PEEK E POKE

La funzione PEEK legge un byte da un particolare indirizzo di memoria. La funzione POKE scrive un valore (espresso in byte) all'indirizzo di memoria specificato.

```
PEEK(ind)        ' ind è uno spiazzamento nel segmento corrente.

POKE ind, byte   ' byte è un valore da 0 a 255.
```

PEEK e POKE generalmente sono riservate per operazioni a basso livello in cui i valori devono essere letti o scritti direttamente in specifici indirizzi di memoria. Sia in PEEK sia in POKE, l'argomento *ind* è uno spiazzamento nel segmento corrente, indicato da DEF SEG. La funzione PEEK ritorna un valore da 0 a 255, che è il valore memorizzato in un indirizzo specifico di una locazione della memoria. Quando è possibile, si può usare la funzione CHR$ per trovare il carattere equivalente del valore numerico del byte. Un'istruzione POKE scrive il valore del byte nell'indirizzo di memoria indicato. Un'istruzione POKE sovrascrive il contenuto corrente dell'indirizzo in questione; per questo motivo, POKE in un indirizzo sbagliato può causare gravi danni.

Un esempio di PEEK e POKE

Questo semplice programma illustra l'uso di PEEK e POKE. Il programma utilizza la funzione SADD per trovare l'indirizzo di una stringa e la funzione

PEEK per leggere la stringa dalla memoria, carattere per carattere. Poi il programma usa POKE per scrivere nuovi caratteri nella locazione della stringa:

```
CLS
salut$ = "ciao "
iniz% = SADD(salut$)
fin% = iniz% + LEN(salut$) - 1

FOR i% = iniz% TO fin%
   PRINT CHR$(PEEK(i%));
   READ nuoCar%
   POKE i%, nuoCar%
NEXT i%
PRINT
PRINT salut$

DATA 103, 111, 111, 100, 45, 98, 121, 101
```

La stringa originale è memorizzata in una variabile chiamata *salut$*. Un ciclo LOOP chiama la funzione PEEK per leggere il valore iniziale di questa variabile e visualizza il valore sullo schermo. Nello stesso ciclo, un'istruzione POKE scrive una nuova sequenza di caratteri nell'indirizzo della memoria. Quando il ciclo è completo, un'istruzione PRINT visualizza il nuovo valore della variabile.

Compatibilità

Sia Turbo Basic sia BASICA hanno le funzioni PEEK e POKE.

SADD

La funzione SADD fornisce lo spiazzamento nel segmento corrente di un valore stringa memorizzato.

```
SADD(var)    ' L'argomento var è il nome di una variabile della stringa.
```

L'argomento di SADD è una semplice variabile stringa o un elemento di un vettore stringa. (Una variabile stringa a lunghezza fissa non è utilizzabile

come argomento di SADD.) La funzione ritorna un valore da 0 a 65.535, che rappresenta lo spiazzamento della stringa nel segmento dati di QuickBASIC. I valori stringa talvolta sono spostati da un indirizzo della memoria ad un altro, durante l'esecuzione di un programma. Per questo motivo si può fare una chiamata a SADD nel momento in cui si ha effettivamente bisogno dell'indirizzo.

Un esempio di SADD

Il programma riportato alle voci PEEK e POKE contiene un esempio della funzione SADD:

```
salut$ = "ciao "
iniz% = SADD(salut$)
```

In questo programma, SADD ritorna lo spiazzamento di "c", il primo carattere nella stringa stessa.

Compatibilità

Né Turbo Basic né BASICA hanno la funzione SADD.

SETMEM

La funzione SETMEM ritorna le dimensioni attuali dello heap e opzionalmente tenta di aumentarle o diminuirle.

```
SETMEM(0)       ' Ritorna le dimensioni dello heap.

SETMEM(num)     ' Aumenta lo heap se num è positivo e lo diminuisce
                ' se è negativo. Ritorna le nuove dimensioni.
```

Lo heap è l'area della memoria in cui QuickBASIC memorizza i vettori dinamici. Il controllo sulle dimensioni dello heap può essere importante in applicazioni che usano QuickBASIC con altri linguaggi. Ogni chiamata alla funzione SETMEM ritorna le dimensioni attuali dello heap. Quando un

argomento è zero, non viene effettuato nessun cambiamento nelle dimensioni dello heap. Un argomento positivo o negativo cerca di aumentare o diminuire le dimensioni dello heap.

Un esempio di SETMEM

Quest'istruzione visualizza le dimensioni correnti dello heap, senza cercare di cambiarle:

```
PRINT SETMEM(0)
```

Compatibilità

Né Turbo Basic né BASICA hanno la funzione SETMEM.

VARSEG E VARPTR

Le funzioni VARSEG e VARPTR insieme forniscono il completo indirizzo di memoria di una variabile o di un vettore di QuickBASIC. VARSEG ritorna il segmento e VARPTR lo spiazzamento all'interno di quel segmento.

```
VARSEG(var)
VARPTR(var)
```

L'argomento di entrambe le funzioni è il nome di una variabile QuickBASIC che appartiene ad un tipo. La funzione VARSEG ritorna un intero da 0 a 65.535, che rappresenta il segmento in cui la variabile è memorizzata. La funzione VARPTR ritorna un intero (nello stesso intervallo di valori) che rappresenta lo spiazzamento della variabile all'interno del segmento. Se l'argomento è un vettore, bisogna fornire l'elemento iniziale (o gli elementi) del vettore stesso. VARSEG e VARPTR ritornano l'indirizzo dell'elemento specificato. Le variabili talvolta vengono spostate da un indirizzo della memoria all'altro, durante un'esecuzione del programma. Per questo motivo, si eseguono le chiamate a VARSEG e VARPTR proprio nel momento in cui serve effettivamente l'indirizzo della variabile.

Alcuni esempi di VARSEG e VARPTR

Alla voce BSAVE e BLOAD vi sono un paio di programmi che utilizzano VARSEG e VARPTR per determinare l'indirizzo di un vettore. Il vettore memorizza le immagini grafiche che vengono prese dallo schermo con le istruzioni GET. Nel processo di memorizzazione di queste immagini su disco (BSAVE) o di lettura di esse dal disco (BLOAD), le funzioni VARSEG e VARPTR forniscono l'indirizzo del vettore. Ad esempio, si consideri questo frammento del programma che crea il file:

```
DIM immFac(1828)
Faccia sorridente
GET (324, 164)-(406, 246), immFac
Faccia triste
GET (324, 164)-(406, 246), immFac(915)

DEF SEG = VARSEG(immFac(0))
BSAVE "facc.mem", VARPTR(immFac(0)), 7312
```

Il vettore in questione si chiama *immFac*. Un'istruzione DEF SEG stabilisce il segmento in cui si trova il vettore, fornito dalla funzione VARSEG, come segmento corrente. Poi la funzione VARPTR dà lo spiazzamento nel segmento. La chiamata a VARPTR si ha nell'istruzione BSAVE che effettivamente salva il contenuto del vettore su disco. Si possono trovare ulteriori informazioni riguardo questo esempio alle voci (grafiche) GET e PUT e in DEF SEG.

Compatibilità

Turbo Basic ha sia la funzione VARSEG sia VARPTR. BASICA ha solo VARPTR, che ritorna uno spiazzamento nel segmento dati BASICA.

Capitolo 27

Chiamate di sistema del DOS e chiamate ad altri linguaggi

Questo capitolo descrive brevemente le varie istruzioni CALL che QuickBASIC fornisce per chiamare le routine scritte in altri linguaggi e per eseguire delle chiamate di sistema del DOS. Per entrambi questi tipi di chiamate, QuickBASIC ha due distinte serie di istruzioni, una che rappresenta lo stile migliore nella programmazione QuickBASIC ed una che dà la compatibilità con le versioni precedenti di BASIC:

- Le istruzioni CALL o CALLS eseguono delle chiamate alle routine scritte usando altri linguaggi, comprese le routine in linguaggio macchina. L'istruzione CALL ABSOLUTE consente la compatibilità con l'istruzione equivalente di BASICA.
- L'istruzione CALL INTERRUPT o CALL INTERRUPTX esegue delle chiamate di sistema DOS. Per compatibilità con le versioni precedenti, QuickBASIC fornisce anche l'istruzione CALL INT86OLD o CALL INT86XOLD.

CALL E CALLS (PROCEDURE IN ALTRI LINGUAGGI)

L'istruzione CALL (o CALLS) esegue una chiamata ad una routine scritta in un linguaggio diverso da QuickBASIC ed invia gli argomenti alla procedura per valore, per indirizzo intrasegmento o intersegmento.

```
CALL Proc (arg)      ' arg è opzionale.

ArgProc              ' sintassi alternativa; è richiesta
                     ' l'istruzione DECLARE.

CALLS Proc (arg)     ' arg è opzionale.
```

Nella sintassi di queste istruzioni, *Proc* è il nome di una routine scritta in un altro linguaggio e *arg* è una lista di valori dell'argomento separati da virgole. Come nelle chiamate alle procedure QuickBASIC, si possono omettere la parola chiave CALL e le parentesi attorno alla lista, purché il programma contenga un'istruzione DECLARE per la procedura. Per default, ogni valore dell'argomento viene inviato per *indirizzo intrasegmento*, cioè solo come spiazzamento all'interno del segmento codice corrente. Per cambiare questo default si può includere nella lista una delle due speciali parole chiave prima dell'argomento:

```
BYVAL arg
SEG arg
```

BYVAL invia alla routine il valore attuale dell'argomento, non un indirizzo. SEG invia il valore come un *indirizzo intersegmento*, che comprende cioè sia il segmento sia lo spiazzamento all'interno di questo. Un vettore è seguito da due parentesi vuote. L'istruzione CALLS invia tutti gli argomenti per indirizzo intersegmento. La parola chiave SEG non è né richiesta né corretta nella lista degli argomenti di questa istruzione.

Un esempio di CALL

Questa istruzione chiama la routine *TestMem* inviandole un argomento per indirizzo intersegmento e uno per valore:

```
CALL TestMem (SEG loc1, BYVAL loc2)
```

Compatibilità

In Turbo Basic, l'istruzione CALL può chiamare un programma del linguaggio macchina *in linea*. Un programma di questo tipo è scritto come una

procedura SUB, con una sequenza di metacomandi $INLINE che precedono ogni linea del codice del programma. In BASICA l'istruzione CALL chiama una routine in linguaggio macchina. La sintassi dell'istruzione consente di utilizzare una variabile numerica che indica il punto di inizio della routine in memoria. Questo indirizzo è uno spiazzamento nel segmento corrente. (Vedere la voce CALL ABSOLUTE per ulteriori dettagli.)

CALL ABSOLUTE

L'istruzione CALL ABSOLUTE chiama una routine in linguaggio macchina, in uno stile compatibile con le chiamate del linguaggio macchina di BASICA.

```
CALL ABSOLUTE (arg, ind)     ' arg è opzionale.
```

La lista opzionale degli argomenti passa ogni argomento alla routine per indirizzo intrasegmento, cioè tramite il solo spiazzamento. L'argomento *ind* richiesto è l'indirizzo di memoria della routine in linguaggio macchina. Questo indirizzo è uno spiazzamento del segmento corrente, designato dall'istruzione DEF SEG. Per un buon utilizzo di CALL ABSOLUTE, bisogna caricare una libreria che definisca ABSOLUTE. La procedura ABSOLUTE è definita nella libreria Quick di default, QB.QLB. Per caricare questa libreria nell'ambiente, basta avviare QuickBASIC come segue:

```
C>QB /L
```

L'istruzione CALL ABSOLUTE è simile all'istruzione CALL in BASICA. Il modo più comodo per gestire in QuickBASIC le chiamate del linguaggio macchina è però quello di utilizzare l'istruzione CALL o CALLS.

Un esempio di CALL ABSOLUTE

La seguente istruzione chiama una routine in linguaggio macchina che è stata caricata in memoria nell'indirizzo dello spiazzamento *ind1*:

```
CALL ABSOLUTE (v1, v2, ind1)
```

Compatibilità

L'istruzione Turbo Basic CALL ABSOLUTE ha una sintassi leggermente diversa:

```
CALL ABSOLUTE ind (arg)
```

Allo stesso modo, l'istruzione CALL in BASICA appare come:

```
CALL ind (arg)
```

In entrambi i casi, *ind* è uno spiazzamento nel segmento corrente designato dall'istruzione DEF SEG.

CALL INT86OLD E CALL INT86XOLD

L'istruzione CALL INT86OLD o CALL INT86XOLD esegue una chiamata di sistema del DOS da QuickBASIC e fornisce dei vettori che contengono i valori dei registri prima e dopo la chiamata.

```
CALL INT86OLD (inter, vett1(), vett2())

CALL INT86XOLD (inter, vett1(), vett2())
```

L'argomento *inter* è un numero di interrupt del DOS da 0 a 255. Gli argomenti *vett1()* e *vett2()* sono entrambi vettori interi progettati per memorizzare i valori dei registri delle CPU. I vettori per l'istruzione CALL INT86OLD contengono ognuno otto elementi interi, che rappresentano rispettivamente i registri AX, BX, CX, DX, BP, SI, DI e FLAGS. I vettori per CALL INT86XOLD contengono due ulteriori elementi, che rappresentano DS e ES. Prima di usare l'istruzione CALL INT86OLD o CALL INT86XOLD, un programma di solito esegue le seguenti operazioni:

1. Usa un metacomando $INCLUDE per incorporare il file QB.BI nel codice sorgente. Questo file contiene le istruzioni DECLARE per le procedure INT86OLD e INT86XOLD.
2. Dichiara *vett1()* e *vett2()* in un'istruzione DIM.

3. Assegna i valori opportuni agli elementi del vettore *vett1*. Questi sono i valori che saranno posti nei registri quando l'interrupt DOS sarà eseguito.

Il vettore *vett2* conterrà la serie completa dei valori dei registri dopo l'interrupt. Il programma sarà in grado di leggere i valori eventualmente modificati direttamente dagli elementi di questo vettore. Per un corretto utilizzo di CALL INT86OLD o CALL INT86XOLD, si deve caricare una libreria che definisca queste procedure. CALL INT86OLD e CALL INT86XOLD vengono definite nella Libreria Quick di default, QB.QLB. Per caricare questa libreria nell'ambiente, avviare QuickBASIC come segue:

```
C>QB /L
```

Il modo migliore per eseguire le chiamate di sistema del DOS in QuickBASIC è comunque quello di utilizzare l'istruzione CALL INTERRUPT o CALL INTERRUPTX.

Compatibilità

Né Turbo Basic né BASICA hanno l'istruzione CALL INT86OLD.

CALL INTERRUPT E CALL INTERRUPTX

L'istruzione CALL INTERRUPT o CALL INTERRUPTX esegue una chiamata di sistema del DOS da QuickBASIC e fornisce delle variabili record che contengono i valori dei registri prima e dopo la chiamata.

```
CALL INTERRUPT (inter, inreg, outreg)
```

```
CALL INTERRUPTX (inter, inreg, outreg)
```

L'argomento *inter* è un numero di interrupt del DOS da 0 a 255. Gli argomenti *inreg* e *outreg* sono entrambi variabili dei record che appartengono al tipo *RegTip* definito dall'utente. Questo tipo, definito nel file sorgente chiamato QB.BI, contiene singoli campi di tipo intero per ognuno dei registri CPU. Gli elementi per l'istruzione CALL INTERRUPT sono *ax, bx,*

cx, dx, bp, si, di e *flags*. Un record esteso, chiamato *RegTipX*, viene definito per l'istruzione CALL INTERRUPTX. Questo tipo di record contiene tutti i campi *RegTipX*, più *ds* e *es*. Prima di usare l'istruzione CALL INTERRUPT o CALL INTERRUPTX, un programma di solito esegue i seguenti compiti:

1. Usa un metacomando $INCLUDE per incorporare il file QB.BI nel codice sorgente. Questo file contiene le dichiarazioni TYPE per i tipi record *RegTip* e *RegTipX* e le istruzioni DECLARE per le procedure INTERRUPT e INTERRUPTX.
2. Usa un'istruzione DIM (o un'altra dichiarazione) per dichiarare le variabili *inreg* e *outreg* come appartenenti al tipo record *RegTip* o *RegTipX*.
3. Assegna gli opportuni valori nei registri nella variabile *inreg*. Questi sono i valori che verranno posti nei registri quando l'interrupt del DOS sarà eseguito.

La variabile record *outreg* conterrà la serie completa dei valori dei registri dopo l'interrupt. Il programma potrà leggere ogni valore modificato direttamente dai campi di questo record. Le procedure INTERRUPT e INTERRUPTX vengono definite nella Libreria Quick di default, QB.QLB. Per caricare questa libreria nell'ambiente, avviare QuickBASIC come segue:

```
C>QB /L
```

Un esempio di CALL INTERRUPT

Questa linea esegue un interrupt del DOS:

```
CALL INTERRUPT(intNum%, inizReg, finReg)
```

Prima di questa chiamata, il programma definisce le variabili *inizReg* e *finReg* come record di *RegTip* ed assegna i valori da porre nei registri macchina ai campi del record *inizReg*.

Compatibilità

L'istruzione CALL INTERRUPT di Turbo Basic ha una sintassi più semplice:

```
CALL INTERRUPT inter
```

In Turbo Basic, i valori iniziali dei registri sono attribuiti dall'istruzione REG. Dopo l'interrupt, un programma può leggere i valori dei registri tramite la funzione REG. BASICA non ha l'istruzione CALL INTERRUPT.

DECLARE (PROCEDURE IN ALTRI LINGUAGGI)

L'istruzione DECLARE dichiara il nome e i parametri della routine scritta in un altro linguaggio.

```
DECLARE SUB Proc CDECL ALIAS "alias" (par)
DECLARE FUNCTION Proc CDECL ALIAS "alias" (par)

' Le clausole CDECL e ALIAS sono opzionali come lo è la
' lista del parametro.
```

Nella sintassi di queste istruzioni, *Proc* è il nome di una routine in un altro linguaggio e *par* è una lista di parametri. La parola chiave CDECL è richiesta se la routine esterna è scritta in C e specifica che gli argomenti sono passati alla routine da destra a sinistra. La clausola ALIAS fornisce il nome alternativo della routine nel file oggetto. Per default, ogni parametro è inviato per *indirizzo intrasegmento*, cioè come solo spiazzamento. Per cambiare questo default, si può inserire una delle due parole chiave che seguono prima di ogni parametro nella lista:

```
BYVAL arg
SEG arg
```

BYVAL invia il valore attuale del parametro, non un indirizzo. SEG invia il parametro come un *indirizzo intersegmento*, che comprende sia il segmento sia lo spiazzamento all'interno di questo. Un parametro vettore è

seguito da due parentesi vuote. Ogni parametro può anche essere seguito da una clausola AS che indica il suo tipo di dati:

```
AS tip
```

Il *tip* delle variabili BYVAL può essere dichiarato come INTEGER, LONG, SINGLE o DOUBLE. Il *tip* di una variabile SEG può essere INTEGER, LONG, SINGLE o DOUBLE, STRING o ANY. La parola chiave ANY inibisce il controllo del tipo.

Un esempio di DECLARE

Questa istruzione dichiara la routine ed i suoi parametri:

```
DECLARE TestMem (SEG v1 AS INTEGER, BYVAL v2 AS INTEGER)
```

Compatibilità

Né Turbo Basic né BASICA hanno l'istruzione DECLARE.

Parte VIII

Esempi di programmi

I cinque programmi descritti e listati nella Parte VIII sono progettati per illustrare i principali argomenti trattati in questo libro:

- Il programma *Compleanno* (Capitolo 28) è una semplice applicazione delle tecniche di gestione di archivi che indica l'utilizzo delle costanti simboliche, delle variabili e delle strutture dati in QuickBASIC.
- Il programma *Mese* (Capitolo 29) visualizza sullo schermo il calendario di uno o più mesi o lo invia alla stampante. I mesi in questione vanno indicati al prompt del DOS, quando si avvia il programma. Questa applicazione presenta gli elementi della programmazione strutturata in QuickBASIC, compresi cicli, procedure e selettori.
- Il programma *Agenda* (Capitolo 30) gestisce un database del calendario per registrare gli appuntamenti giornalieri. Il programma serve come introduzione a varie tecniche d'input ed output di QuickBASIC, compresi l'input da tastiera, l'output sullo schermo e la programmazione di file di dati con file ad accesso casuale e random.
- Il programma *GestioneDate* (Capitolo 31) è una semplice interfaccia utente per il programma *Agenda*. Consente all'utente di selezionare una data passata, presente o futura per visualizzare o modificare gli appuntamenti giornalieri. *GestioneDate* illustra le istruzioni di gestione degli eventi di QuickBASIC.

- Il programma *DateStoriche* (Capitolo 32) è un gioco a quiz che prova l'abilità del giocatore nel ricordare le date storiche. Il programma mostra l'utilizzo di molti strumenti grafici QuickBASIC ed altre tecniche, compresa la produzione di suoni.

Ogni capitolo della Parte VIII è formato da tre parti: la prima spiega nei dettagli ciò che il programma esegue, come avviarlo e descrive le prerogative per utilizzarlo; la seconda presenta brevemente la struttura del programma stesso e l'ultima, infine, ne dà un listato completo.

Capitolo 28

Il programma Compleanno

Il programma *Compleanno* è progettato per dimostrare l'utilizzo delle costanti simboliche, delle variabili, dei vettori, delle strutture record definite dall'utente, nonché di molti altri argomenti trattati nella Parte I di questo libro.

ATTIVAZIONE DEL PROGRAMMA

Compleanno lavora con l'archivio degli impiegati di una immaginaria società. L'archivio, che è memorizzato nell'istruzione DATA collocata nel listato del programma, contiene le seguenti informazioni su ogni dipendente:

1. Nome e cognome
2. Qualifica
3. Data di assunzione
4. Data di nascita

Ecco un esempio di record preso dall'archivio:

```
'     Nome         Qualifica          Data Assun.    Data Nasc.
DATA  Billi, Anna,  Segretaria,        8, 1, 1988,    6, 4, 1966
```

Si noti che ognuna delle due date è espressa come una sequenza di tre interi separati da virgole. Partendo da questo archivio, il programma genera tabelle ordinate di informazioni che riflettono il budget della ditta per le gratifiche annuali dei dipendenti. Tutti gli anni ogni dipendente, il giorno del suo compleanno, riceve dalla ditta una gratifica che è uguale a L. 50.000 per ogni anno di anzianità. Il programma *Compleanno* usa l'archivio dei dipendenti per ottenere molte informazioni importanti:

1. Il numero di anni che ogni dipendente ha trascorso alle dipendenze della ditta.
2. L'ammontare della gratifica che ogni dipendente percepirà per l'anno corrente.
3. L'età di ogni dipendente alla fine dell'anno in corso.
4. La data del compleanno di ogni dipendente e il giorno della settimana in cui esso cadrà nell'anno in corso.

Oltre a queste informazioni su ogni impiegato, il programma calcola anche la somma totale delle gratifiche che la ditta darà nell'anno in corso. Perché questi dati siano accessibili in diverse situazioni, il programma *Compleanno* è in grado di visualizzare la tabella secondo diversi ordini. In particolare, quando si avvia il programma, sullo schermo appare il seguente menù:

```
Gratifiche di compleanno dei dipendenti per il 1990.
Organizzare la tabella in base a:

N)omi.
Q)ualifica.
A)nzianità (ordine ascendente).
D)iscendente (Anzianità).
E)tà (dei dipendenti).
C)ompleanni.

   –>
```

Si può selezionare uno di questi sei metodi di classificazione. Per visualizzare la tabella secondo un particolare ordine, basta semplicemente premere una sola lettera che rappresenta una selezione del menù: N, Q, A, D, E o C.

Ad esempio, ecco la videata della tabella secondo il metodo di classificazione in base ai nomi dei dipendenti:

Gratifiche di compleanno dei dipendenti per il 1990.

Nome	Qualifica	Anzianità	Gratifica	Età	Compleanno
Billi, Anna,	Segretaria,	2,9	100.000	24	Ven., Apr. 6
Rossi, Paolo,	Impiegato,	5,7	250.000	35	Sab., Lug. 7
Poli, Luca,	Ragioniere,	3,8	150.000	31	Lun., Gen. 1
Aimi, Carla,	Impiegata,	7,5	350.000	40	Mer., Set. 5
Telli, Rosa,	Dirigente,	7,8	350.000	38	Dom., Set. 9
Dadda, Marco,	Dir. Vend.,	1,7	50.000	44	Dom., Mar. 4
Frosi, Mara,	Rappresent.,	2,6	100.000	25	Dom., Ago. 5
Mari, Diego,	Programmat.,	1,8	50.000	24	Mer., Ago. 8
Vago, Mario,	Ricercatore,	2,5	100.000	27	Ven., Set. 7
Pini, Clara,	Segretaria,	3,1	150.000	26	Ven., Feb. 2
Orio, Tina,	Rappresent.,	3,1	150.000	31	Gio., Mag. 3
Gosi, Milo,	Programmat.,	4,4	200.000	22	Gio., Mar. 1
Sora, Fabio,	Dirigente,	6,1	300.000	40	Mer., Giu. 6
Radi, Lara,	Impiegata,	2,6	100.000	30	Gio., Gen. 4
Bosio, Carlo,	Dir. Tecn.,	3,9	150.000	35	Mer., Mar. 7

Totale delle gratifiche 1990 per 15 impiegati: L.2.550.000

Nella parte inferiore di ogni schermata, il programma visualizza questa domanda:

Visualizzo un'altra tabella? <S> o <N>

Se si risponde premendo il tasto S, il programma ripete il menù dei metodi di classificazione dando l'opportunità di selezionarne uno nuovo per la visualizzazione della tabella dei dipendenti. Ad esempio, ecco un'ulteriore videata della tabella ordinata in base ai mesi di compleanno dei dipendenti:

Nome	Qualifica	Anzianità	Gratifica	Età	Compleanno
Poli, Luca,	Ragioniere,	3,8	150.000	31	Lun., Gen. 1
Radi, Lara,	Impiegata,	2,6	100.000	30	Gio., Gen. 4
Pini, Clara,	Segretaria,	3,1	150.000	26	Ven., Feb. 2
Gosi, Milo,	Programmat.,	4,4	200.000	22	Gio., Mar. 1
Dadda, Marco,	Dir. Vend.,	1,7	50.000	44	Dom., Mar. 4
Bosio, Carlo,	Dir. Tecn.,	3,9	150.000	35	Mer., Mar. 7
Billi, Anna,	Segretaria,	2,9	100.000	24	Ven., Apr. 6
Orio, Tina,	Rappresent.,	3,1	150.000	31	Gio., Mag. 3

```
Sora, Fabio,  Dirigente,        6,1       300.000      40      Mer., Giu. 6
Rossi, Paolo, Impiegato,        5,7       250.000      35      Sab., Lug. 7
Frosi, Mara,  Rappresent.,      2,6       100.000      25      Dom., Ago. 5
Mari, Diego,  Programmat.,      1,8        50.000      24      Mer., Ago. 8
Aimi, Carla,  Impiegata,        7,5       350.000      40      Mer., Set. 5
Vago, Mario,  Ricercatore,      2,5       100.000      27      Ven., Set. 7
Telli, Rosa,  Dirigente,        7,8       350.000      38      Dom., Set. 9

Totale delle gratifiche 1990 per 15 impiegati:     L.2.550.000
```

Per terminare il programma, premere il tasto N in risposta alla domanda *"Visualizzo un'altra tabella?"* quando appare nella parte inferiore dello schermo.

LA STRUTTURA DEL PROGRAMMA

Una delle caratteristiche del progetto di questo programma è che l'archivio si può ritenere relativamente stabile, cioè che nel tempo ci sono solo lievi cambiamenti all'interno del gruppo dei dipendenti della ditta. È possibile aggiungere, cancellare o modificare un record solo riadattando appropriatamente le istruzioni DATA e poi ricompilando il programma. (Seguendo questo stesso approccio, si può usare il programma *Compleanno* per ogni archivio di dipendenti che si immette nelle linee DATA. Si noti che vi è anche un'istruzione DATA che specifica il numero dei record nell'archivio e una costante simbolica, GRATIFICANNUALE, che rappresenta la somma base della gratifica. Entrambi questi valori possono essere adattati per altri contesti.) All'inizio del listato del programma vi sono delle dichiarazioni per le strutture dati e costanti simboliche, molte delle quali vengono presentate come esempi nelle voci della Parte I. In particolare, il programma dichiara un tipo record chiamato *dip* e un vettore dinamico composto di record *dip* e chiamato *staff*, che memorizza l'intero archivio degli impiegati durante un'esecuzione del programma. Inoltre, il programma crea tre vettori statici per memorizzare specifiche informazioni relative al calendario: *nomeMese$* per i nomi abbreviati dei mesi; *giorniMese$* per il numero di giorni in ogni mese e *giorniSett$* per i nomi abbreviati dei giorni della settimana. La parte eseguibile del programma principale inizia con le chiamate a due sottoprogrammi che leggono le informazioni dalle istruzioni DATA: *LeggiInfCal* legge i dati nei vettori *nomeMese$*, *giorniMese$* e *giorniSett$*. Poi *LeggiRecStaff* legge l'archivio

impiegati nel vettore *staff* e attribuisce anche i valori ad alcuni campi calcolati. *LeggiRecStaff* esegue delle chiamate alle tre funzioni progettate per ricavare i dati sugli impiegati: *CalcEtà%* calcola l'età di un dipendente, la funzione *CalcAnz%* calcola il numero di anni che ogni dipendente ha trascorso alle dipendenze della ditta e *CalcGior%* dà un intero da 1 a 366, che rappresenta il numero del giorno di compleanno del dipendente. (Il programma usa questo campo finale anche per ordinare l'archivio in base ai compleanni.) Queste tre funzioni, a turno, si servono di altre funzioni del calendario, con compiti più generali, che fanno sempre parte del programma. Per analizzare più approfonditamente queste funzioni, basta esaminare il programma *Mese*, presentato nel Capitolo 29. Dopo queste fasi preliminari necessarie per creare le strutture dati, un singolo ciclo DO controlla il funzionamento del programma principale:

```
DO
   CLS
   PRINT TITLE; RIGHT$(DATE$, 4); "."

   OrdinStaff Selez%

   VisualTab

LOOP UNTIL Fine%
```

La funzione *Selez%* presenta sullo schermo il menù del programma e fa sì che l'utente selezioni il metodo di classificazione. Questa scelta viene poi passata come un intero al sottoprogramma *OrdinStaff*, che effettivamente esegue la scelta. Si noti l'aspetto compatto di queste due chiamate di una procedura:

```
OrdinStaff Selez%
```

In risposta a questa istruzione, QuickBASIC prima esegue la chiamata alla funzione *Selez%* e poi invia il valore di ritorno da questa funzione come un argomento al sottoprogramma *OrdinStaff*. Una volta che il programma ha ordinato l'archivio, una chiamata al sottoprogramma *VisualTab* visualizza le informazioni sullo schermo. Infine, in fondo al ciclo DO, una chiamata alla funzione *Fine%* richiede all'utente una risposta affermativa o negativa a questa domanda:

```
Visualizzo un'altra tabella? <S> o <N>
```

Se la risposta dell'utente è affermativa, la funzione *Fine%* ritorna un valore logico falso ed il ciclo continua, ma quando l'utente infine preme il tasto N come risposta, *Fine%* dà un valore logico vero e l'esecuzione finisce.

IL LISTATO DEL PROGRAMMA COMPLEANNO

```
' COMPLEANNO.BAS
' Crea una tabella delle gratifiche di compleanno per i
' dipendenti di una piccola ditta (immaginaria).

' Il proprietario dell'azienda dà ogni anno ai suoi
' dipendenti una gratifica di compleanno, il cui valore è
' calcolato moltiplicando L.50.000 per ogni anno di
' attività svolto nella ditta fino alla fine dell'anno in
' corso.

' Il listato memorizza "l'archivio" impiegati in una
' sequenza di istruzioni DATA, collocate alla fine del
' programma principale. Il programma legge l'archivio in
' un vettore di record e poi offre all'utente molti
' schemi di classificazione possibili per la tabella
' degli output: la tabella dei dipendenti può essere
' ordinata secondo i nomi, la descrizione dei lavori,
' l'anzianità, l'età o i compleanni ordinati, da gennaio
' a dicembre.

DECLARE SUB LeggiInfCal ()
DECLARE SUB LeggiRecStaff ()
DECLARE SUB OrdinStaff (selezTast%)
DECLARE SUB VisualTab ()
DECLARE FUNCTION CalcEtà% (me%, gio%, an%)
DECLARE FUNCTION CalcGior% (me%, gio%)
DECLARE FUNCTION CalcAnz% (me%, gio%, an%)
DECLARE FUNCTION NumGior$ (mese%, giorno%, anno%)
DECLARE FUNCTION NumData& (mese%, giorno%, anno%)
DECLARE FUNCTION Selez% ()
DECLARE FUNCTION AnnoBisest% (anno%)
DECLARE FUNCTION GiorniMese% (mese%, anno%)
DECLARE FUNCTION Fine% ()

CONST FALSE = 0
CONST TRUE = NOT FALSE

CONST ORDINENOM = 1     ' Ordina secondo il nome degli impiegati.
CONST ORDINEQUAL = 2    ' Ordina secondo il tipo di  lavoro.
```

```
CONST ORDINEANZA = 3    ' Ordina secondo gli anni di anzianità,
                        ' ascendente.
CONST ORDINEANZD = 4    ' Ordina secondo gli anni di anzianità,
                        ' discendente.
CONST ORDINETA = 5      ' Ordina secondo l'età.
CONST ORDINECOMP = 6    ' Ordina secondo i compleanni.

CONST GRATIFICANNUALE = 50
CONST TITLE = "Gratifiche di Compleanno dei Dipendenti per "

' Definisce il tipo record per l'archivio impiegati.

TYPE dip
   Cogn AS STRING * 9      ' Cognome del dipendente.
   Nome  AS STRING * 9     ' Nome del dipendente.
   qualif AS STRING * 18   ' Qualifica del dipendente.
   anniAnz AS SINGLE       ' Anni di anzianità.
   etàAtt AS INTEGER       ' Età del dipendente.
   meseNasc AS INTEGER     ' Mese di nascita.
   compl AS INTEGER        ' Giorno di nascita.
   giorAnno AS INTEGER     ' Intero che rappresenta il giorno di
                           ' compleanno nel calendario.
END TYPE

COMMON SHARED dimStaff%

CLS

' Creare dei vettori per memorizzare le informazioni del calendario.

DIM SHARED nomiMese%(12), giorniMese%(12), giorniSett$(7)
LeggiInfCal

' Crea l'archivio degli impiegati e lo memorizza nel
' vettore chiamato staff.

READ dimStaff%
DIM SHARED staff(dimStaff%) AS dip
LeggiRecStaff

DO
   CLS
   PRINT TITLE; RIGHT$(DATE$, 4); "."

' Ordina l'archivio degli impiegati con uno dei cinque tasti,
' selezionati attraverso una chiamata alla funzione Selez%.

   OrdinStaff Selez%

' Visualizza la tabella delle gratifiche dei dipendenti.
```

```
   VisualTab

LOOP UNTIL Fine%

END

' Informazioni del calendario:

DATA Gen, 31, Feb, 29, Mar, 31, Apr, 30, Mag, 31, Giu, 30
DATA Lug, 31, Ago, 31, Set, 30, Ott, 31, Nov, 30, Dic, 31
DATA Dom, Lun, Mar, Mer, Gio, Ven, Sab

DATA 15 : REM Numero attuale di dipendenti.

Archivio:

'     Nome           Qualifica       Data Assun.     Data Nasc.

DATA  Billi, Anna,   Segretaria,     8, 1, 1988,     6, 4, 1966
DATA  Rossi, Paolo, Impiegato,       7, 3, 1985,     7, 7, 1955
DATA  Poli, Luca,    Ragioniere,     1, 2, 1987,     1, 1, 1959
DATA  Aimi, Carla,   Impiegata,      3, 5, 1983,     5, 9, 1950
DATA  Telli, Rosa,   Dirigente,      5, 2, 1983,     9, 9, 1952
DATA  Dadda, Marco, Dir. Vend.,      1, 3, 1989,     4, 3, 1946
DATA  Frosi, Mara,   Rappresent.,    2, 4, 1988,     5, 8, 1965
DATA  Mari, Diego,   Programmat.,    6, 2. 1989,     8, 8, 1966
DATA  Vago, Mario,   Ricercatore,    2, 5, 1988,     7, 9, 1963
DATA  Pini, Clara,   Segretaria,     1, 9, 1987,     2, 2, 1964
DATA  Orio, Tina,    Rappresent.,    1, 9, 1987,     3, 5, 1959
DATA  Gosi, Milo,    Programmat.,    4, 6, 1986,     1, 3, 1968
DATA  Sora, Fabio,   Dirigente,      8, 9, 1984,     6, 6, 1950
DATA  Radi, Lara,    Impiegata,      7, 4, 1988,     4, 1, 1960
DATA  Bosio, Carlo, Dir. Tecn.,      9, 1, 1987,     7, 3, 1955

FUNCTION CalcEtà% (me%, gio%, an%)

' La funzione CalcEtà% calcola gli anni che un dipendente
' compirà nell'anno in corso.

' Chiama NumData& per rappresentare con un intero lungo la data di
' nascita del dipendente e il compleanno nell'anno in corso.

   dataNasc& = NumData&(me%, gio%, an%)
   annoCorr& = NumData&(me%, gio%, VAL(RIGHT$(DATE$, 4)))

' Calcola l'età attuale del dipendente.

   CalcEtà% = CINT((annoCorr& - dataNasc&) / 365)

END FUNCTION
```

```
FUNCTION CalcGio% (me%, gio%)

' La funzione CalcGio% ritorna un intero da 1 a 366 che
'     rappresenta la data del calendario (cioè, il mese ed
'     il giorno) inviata come argomenti. Questa funzione
'     utilizza il vettore globale chiamato giorMese%, che
'     è creato a livello del programma principale.

   temp% = 0
   FOR i% = 1 TO me% - 1
      temp% = temp% + giorMese%(i%)
   NEXT i%

   CalcGio% = temp% + gio%

END FUNCTION

FUNCTION CalcAnz (me%, gio%, an%)

' La funzione CalcAnz calcola alla fine dell'anno in
'     corso il numero degli anni che un determinato
'     impiegato ha trascorso alle dipendenze della ditta.
'     La funzione ritorna un numero in precisione
'     semplice.

   iniz$ = NumData&(me%, gio%, an%)
   annoCorr& = NumData&(12, 31, VAL(RIGHT$(DATE$, 4)))

   CalcAnz = (annoCorr& - iniz&) / 365

END FUNCTION

FUNCTION NumData& (mese%, giorno%, anno%)

' La funzione NumData& ritorna un intero lungo che
'     rappresenta la data specificata negli argomenti. Il
'     valore pari a 1 rappresenta il 1° Gennaio 1900.

   annoIniz% = 1900
   Genn% = 1
   GiorPerAn& = 365

' Se la funzione riceve una data non valida,
' il valore di ritorno è zero.

   tropPr% = (anno% < annoIniz%)
   meseErr% = (mese% < 1 OR mese% > 12)
   giorErr% = (gio% < 1 OR gio% > GiorniMese%(mese%, anno%))
   IF tropPr% OR meseErr% OR giorErr% THEN
      NumData& = 0
```

```
      EXIT FUNCTION
   END IF

' Altrimenti, calcola i giorni trascorsi dal 1900.

   num& = giorPerAn& * (anno% - annoIniz%)

' Aggiunge giorni extra per gli anni bisestili.

   FOR annoCorr% = annoIniz% TO anno% - 1 STEP 4
      IF AnnoBisest%(annoCorr%) THEN num& = num& + 1
   NEXT annoCorr%

' Conta i giorni dei mesi —
' escludendo il mese specificato.

   FOR MeseCorr% = Genn% TO mese% - 1
      num& = num& + giorniMese%(meseCorr%, anno%)
   NEXT meseCorr%

' Aggiunge i giorni fino alla data specificata e
' ritorna il valore ottenuto.

   num& = num& + gior%
   NumData& = num&

END FUNCTION

FUNCTION NumGior% (mese%, giorno%, anno%)

'  La funzione NumGior% ritorna un valore da 1 a 7,
'     che rappresenta il giorno della settimana (da
'     domenica a sabato) della data specificata negli
'     argomenti. Un valore di ritorno pari a zero
'     corrisponde ad una data non valida.

   g& = NumData&(mese%, giorno%, anno%)
   IF g& <> 0 THEN gds% = g& MOD 7 + 1 ELSE gds% = 0

   NumGior% = gds%

END FUNCTION

FUNCTION Selez%

' La funzione Selez% mette a disposizione dell'utente un
'     menù che propone cinque chiavi di classificazione per
'     l'archivio dei dipendenti e lo invita ad esprimere
'     una scelta. Il valore di ritorno è un intero che va
'     da 1 a 5 e rappresenta una di queste cinque chiavi.
```

```
PRINT "Organizzare la tabella in base a:
PRINT
PRINT "  N)omi."
PRINT "  Q)ualifica."
PRINT "  A)nzianità (ordine ascendente)."
PRINT "  D)iscendente.(Anzianità)."
PRINT "  E)tà (dei dipendenti)."
PRINT "  C)ompleanni."
PRINT "
PRINT "  ->";

LOCATE , , 1

' Cicla fin che l'utente non preme un tasto che
' rappresenta una delle cinque opzioni del menù.

DO
   scelta$ = UCASE$(INKEY$)
   scelta% = INSTR("NQADEC", scelta$)
LOOP WHILE scelta$ = "" OR scelta% < 1
CLS

Selez% = scelta%

END FUNCTION

FUNCTION AnnoBisest% (anno%)

' La funzione AnnoBisest% ritorna un valore booleano che
' indica se un determinato anno è bisestile.

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)

   AnnoBisest% = divPer4% AND (secolo% IMP secolo400%)

END FUNCTION

FUNCTION GiorniMese% (mese, anno%)

' La funzione GiorniMese% ritorna un intero da 28 a 31,
'     che indica il numero di giorni in un particolare
'     mese. L'argomento anno% è richiesto per determinare
'     il numero di giorni di febbraio.

' Ritorna un valore pari a zero per un mese non valido.

IF mese% < 1 OR mese% > 12 THEN
   GiorniMese% = 0
```

```
   EXIT FUNCTION
END IF

' Altrimenti, determinare il numero di giorni del mese.

   giorni% = giorMese%(mese%)
   IF mese% = 2 AND NOT AnnoBisest%(anno%) THEN giorni% = giorni% - 1

   GiorniMese% = giorni%

END FUNCTION

FUNCTION Fine%

' La funzione Fine% si informa se l'utente vuole vedere
' un'altra tabella ordinata. La funzione ritorna un
' valore pari a vero quando l'utente ha finito di
' consultare le tabelle dei dati.

LOCATE 23, 10
PRINT "Visualizzo un'altra tabella? <S> o <N> ";

DO
   risp$ = UCASE$(INKEY$)
LOOP UNTIL risp$ <> "" AND INSTR("SN", risp$) <> 0

Fine% = (risp$ = "N")

END FUNCTION

SUB LeggiInfCal

' Il sottoprogramma LeggiInfCal legge delle informazioni
'     da una serie di istruzioni DATA che si trovano
'     all'inizio del programma: i 12 nomi dei mesi ed i
'     corrispondenti numeri dei giorni dei mesi, nonché i
'     nomi dei 7 giorni. Queste informazioni vengono
'     memorizzate nei tre vettori condivisi chiamati
'     nomiMese$, giorMese% e giorSett$.

FOR i% = 1 TO 12
   READ nomiMese$(i%)
   READ giorMese%(i%)
NEXT i%

FOR i% = 1 TO 7
   READ giorSett%(i%)
NEXT i%

END SUB
```

```
SUB LeggiInfCal

' Il sottoprogramma LeggiInfCal legge l'archivio degli
'     impiegati da una serie di istruzioni DATA che si
'     trovano vicino all'inizio del programma. L'archivio
'     è memorizzato in un vettore condiviso chiamato
'     staff, definito come vettore di record.

FOR i% =1 TO dimStaff%
   READ staff(i%).cogn
   READ staff(i%).nome$
   READ staff(i%).qualif$
   READ meseIniz%
   READ giorIniz%
   READ annoIniz%

' Calcola, il numero degli
' anni che un determinato impiegato ha trascorso alle
' dipendenze della ditta alla fine dell'anno in corso.

   staff(i%).anzian = CalcAnz(meseIniz%, giorIniz%,annoIniz%)

   READ meseNasc%
   READ giorNasc%
   READ annoNasc%

' Calcola gli anni che un dipendente compirà nell'anno in corso.

   staff(i%).etàAtt = CalcEtà%(meseNasc%, giorNasc%, annoNasc%)

' Calcola un intero che rappresenta il mese e il giorno
' del compleanno di un dipendente.

   staff(i%).meseNasc = meseNasc%
   staff(i%).giorNasc = giorNasc%
   staff(i%).giorAnno = CalcGior%(meseNasc%, giorNasc%)

   NEXT i%

END SUB

SUB OrdinStaff (Chiave%)

' Il sottoprogramma OrdinStaff mette in ordine i record
'     memorizzati nel vettore Staff. Questa routine è in
'     grado di fare una classificazione usando come chiave
'     di ricerca uno dei campi.

' Se l'utente ha scelto l'ordine decrescente rispetto
' agli anni passati alle dipendenze della ditta, semplicemente
```

```
' rilegge i dati nel loro ordine originale.

IF Chiave% = ORDINEANZD THEN
   RESTORE Database
   LeggiRecStaff
   EXIT SUB
END IF

' Altrimenti usa una routine di bubble-sort per stabilire
' un nuovo ordine per i record nell'archivio.

FOR i% = 1 TO dimStaff% - 1
   FOR j% = i% + 1 TO dimStaff%
      scamb% = FALSE

' Decide la chiave rispetto a cui ordinare indicata dal
' valore del parametro Chiave%.

SELECT CASE Chiave%

   CASE ORDINENOM
      Nomin1$ = staff(i%).Cogn + staff(i%).Nome
      Nomin2$ = staff(j%).Cogn + staff(j%).Nome
      scamb% = Nomin1$ > Nomin2$

   CASE ORDINEQUAL
      scamb% = staff(i%).qualif > staff(j%).qualif

   CASE ORDINEANZA
      scamb% = staff(i%).Anzian > staff(j%).Anzian

   CASE ORDINETA
      scamb% = staff(i%).etàAtt < staff(j%).etàAtt

   CASE ORDINECOMP
      scamb% = staff(i%).giorAnno > staff(j%).giorAnno

END SELECT

' Scambia di posto una coppia di record solo se dal confronto
' è risultato un valore pari a TRUE per la variabile logica scamb%.

      IF scamb% THEN SWAP staff(i%), staff(j%)

   NEXT j%
NEXT i%

END SUB

SUB VisualTab
```

```
' Il sottoprogramma VisualTab visualizza l'archivio
'    ordinato dei dipendenti.

' Visualizza il titolo e le intestazioni della colonna.

   anno$ = RIGHT$(DATE$, 4)
   PRINT SPACE$(17); TITLE; anno$
   PRINT
   PRINT " Nome del Dipendente Qualifica Anni ";
   PRINT "Gratifica Età Compleanno"
   PRINT " --------- ----   -- ";
   PRINT "--   -  ----"

' Visualizza una linea per ogni record.

   gratifTot% = 0
   FOR i% = 1 TO dimStaff%
      PRINT staff(i%).Cogn; staff(i%).Nome; staff(i%).qualif;
      PRINT USING " ##.# "; staff(i%).Anzian;
      gratif% = INT(staff(i%).Anzian) * BONUSPERYEAR
      gratifTot% = gratifTot% + gratif%
      PRINT USING "$$###.## "; gratif%
      PRINT USING "## "; staff(i%).etàCorr;
      me% = staff(i%).annoNasc
      gio% = staff(i%).giorNasc
      an% = VAL(RIGHT$(DATE$, 4))
      PRINT giorSett$(NumGior%(me%, gio%, an%)); "., ";
      PRINT nomiMese$(me%); ". "; LTRIM$(STR$(gio%))
   NEXT i%

' Visualizza il totale delle gratifiche dell'anno in
' corso per i dipendenti.

   PRINT
   PRINT "Totale "; anno$; " gratifiche per ";
   PRINT USING "## dipendenti:  "; dimStaff%;
   PRINT USING "$$##,###.##"; gratifTot%

END SUB
```

Capitolo 29

Il programma Mese

Il programma *Mese* illustra molte delle istruzioni e strutture esaminate nella Parte II "Struttura del programma e flusso di controllo", compresi i sottoprogrammi, le funzioni e molti cicli e selettori. Il programma è progettato per essere compilato come un programma EXE a funzionamento autonomo e per essere attivato direttamente dal DOS.

ATTIVAZIONE DEL PROGRAMMA

Mese è un semplice strumento che visualizza o stampa i mesi passati, presenti o futuri. Il primo mese che può essere esaminato è gennaio 1900; l'output del suo calendario può estendersi nei secoli successivi. Una volta che si è compilato il programma, si possono richiedere da *Mese* molti diversi tipi di calendari. L'output dipende dalle informazioni della linea di comando che si dà dal prompt del DOS. Per prima cosa, si può specificare un particolare mese chiamando il programma secondo il seguente formato:

```
MESE me, an
```

dove *me* è un intero da 1 a 12, che rappresenta il mese e *an* è un intero a quattro cifre che indica l'anno. Ad esempio, questo comando visualizza sullo schermo marzo 1985:

```
C>MESE 3, 1985
```

Da questo comando risulta la seguente pagina:

```
           Marzo 1985
D    L    M    M    G    V    S
---------------------
                         1    2
 3    4    5    6    7    8    9
10   11   12   13   14   15   16
17   18   19   20   21   22   23
24   25   26   27   28   29   30
31
```

Esaminando questo output si possono vedere alcuni dei compiti che *Mese* deve eseguire. Per prima cosa esso calcola il giorno della settimana con cui il mese in questione inizia, in questo caso venerdì. Determina anche il numero dei giorni del mese e ne riproduce la videata. Si può avviare il programma anche con molti altri formati di comando. Il più semplice è quello di immettere il nome del programma, senza alcun parametro:

```
C>MESE
```

In risposta, il programma visualizza sullo schermo due mesi: quello corrente e il successivo; ad esempio:

```
        Novembre 1990

D    L    M    M    G    V    S
---------------------
                    1    2    3
 4    5    6    7    8    9   10
11   12   13   14   15   16   17
18   19   20   21   22   23   24
25   26   27   28   29   30

        Dicembre 1990

D    L    M    M    G    V    S
---------------------
                              1
 2    3    4    5    6    7    8
 9   10   11   12   13   14   15
16   17   18   19   20   21   22
23   24   25   26   27   28   29
30   31
```

Il programma reagisce nello stesso modo anche se si forniscono dal prompt del DOS parametri non appropriati. Ad esempio, se si immette 15 come il

numero del mese o 1885 come l'anno, il programma riconosce questo input come non valido e visualizza, invece, il mese corrente e quello successivo. Infine, si può attribuire un solo parametro numerico al programma *Mese*. Se il numero è un intero da 1 a 12, *Mese* visualizza il mese specificato dell'anno in corso. Ad esempio, questo comando dà la videata del mese di novembre di quest'anno:

```
C>MESE 11
```

In alternativa, se il singolo parametro numerico è un intero a quattro cifre (maggiore o uguale a 1900), il programma visualizza tutti i dodici mesi dell'anno specificato. Ad esempio, questo comando visualizza i dodici mesi del 1990:

```
C>MESE 1990
```

Se si preferisce che l'output del programma *Mese* sia stampato piuttosto che visualizzato su schermo, basta semplicemente aggiungere i caratteri /P alla fine della linea di comando:

```
C>MESE 3, 1985 /P
```

(Così come lo stesso nome del programma, anche la notazione /P può essere in lettere maiuscole o minuscole.) Quando si include /P, il programma *Mese* visualizza sullo schermo questo messaggio:

```
Premere la barra spaziatrice quando la stampante è pronta.
```

Ciò consente di accendere la stampante e, se necessario, sistemare la carta. Quando si preme la barra spaziatrice, il mese richiesto viene passato alla stampante. Il parametro /P lavora con tutti i formati del programma *Mese*. Infine, si può usare il dispositivo di *ridirezionamento* del DOS per scrivere l'output nel file di testo sul disco. Ad esempio, questo comando dà i dodici mesi del 1990 e li memorizza come un file di testo chiamato CAL1990.TXT:

```
C>MESE 1990 > CAL1990.TXT
```

LA STRUTTURA DEL PROGRAMMA

Il listato di *Mese* contiene commenti estesi che aiutano a capire come lavora il programma. Ecco un breve schema delle varie procedure del programma. Il programma principale (o "codice modulare" che si trova all'inizio del listato) contiene solo un'istruzione eseguibile, una chiamata al sottoprogramma *Output*. *Output* ha il compito di leggere le istruzioni della linea di comando dell'utente e fornire l'output richiesto. Una chiamata alla procedura *LeggiLiCom* fornisce il numero del mese *(me%)*, l'anno *(an%)* e un altro intero *(operaz%)* che rappresenta il tipo di output che l'utente ha richiesto. *Output* poi usa una struttura SELECT CASE per selezionare l'operazione appropriata. Una o più chiamate alla procedura *StampOVisual* comporta la rappresentazione di ogni mese del calendario sullo schermo o alla stampante. *StampOVisual* esegue una chiamata o alla procedura *StampMese* per stampare su carta il mese oppure alla procedura *VisualMese* per visualizzare sullo schermo il mese. Entrambe queste procedure utilizzano tre delle funzioni del programma per fornire informazioni sul mese in questione: la funzione *NumGior%* ritorna un intero da 1 a 7 che rappresenta il giorno della settimana da cui inizia il mese; *GiorniMese%* ritorna un intero da 28 a 31 che è il numero di giorni del mese e *NomeMese%* ritorna una stringa che rappresenta il nome del mese. La funzione *NumGior%* chiama la funzione *NumData&* per dare un intero che rappresenta il primo giorno del mese in questione. Questo valore, un intero lungo, è il conteggio del numero di giorni dal 1 gennaio 1900 alla data scelta. Dato questo intero, la funzione *NumGior%* ha bisogno solo di eseguire una semplice operazione MOD per determinare il giorno d'inizio di ogni mese. (La funzione *NumData&*, come molti altri strumenti sviluppati in questo programma, è presente ancora in altri progetti di programmazione presentati in questa Parte.)

IL LISTATO DEL PROGRAMMA MESE

```
' MESE.BAS
' Visualizza uno o più mesi del calendario.

' Il programma legge le istruzioni della linea di comando
' nella forma:

'        MESE me, an
```

```
'  dove me è un intero da 1 a 12 e an è un anno a quattro cifre,
'  maggiore o uguale a 1900. Si può anche fornire
'  un solo intero per richiedere un mese dell'anno in
'  corso:

'     MESE me

'  Oppure si può usare un intero a quattro cifre per
'  avere un anno intero dei mesi del calendario:

'     MESE anno

'  Se si omettono le istruzioni della linea di comando —
'  o se esse non sono valide — il programma visualizza il
'  mese corrente e quello successivo.

'  Inoltre, si può includere /P alla fine della linea di
'  comando per inviare la videata del calendario alla
'  stampante piuttosto che allo schermo.

DECLARE SUB Output()
DECLARE SUB LeggiLiCom (qualeOutput%, mese%, anno%)
DECLARE SUB PrepStamp ()
DECLARE SUB StampaMese (mese%, anno%)
DECLARE SUB StampOVisual (st%, me%, an%)
DECLARE SUB VisualMese (mese%, anno%)
DECLARE FUNCTION NumGior% (mese%, giorno%, anno%)
DECLARE FUNCTION NumData& (mese%, giorno%, anno%)
DECLARE FUNCTION AnnoBisest% (anno%)
DECLARE FUNCTION Mese$ (numMese%)
DECLARE FUNCTION GiorniMese% (mese%, anno%)

'  Definire le costanti per rappresentare le varie opzioni
'  del programma.

CONST commErr% = 0           ' La stringa COMMAND$ dell'utente
                             ' non è valida.
CONST nessComm% = 1          ' L'utente non fornisce nessuna
                             ' stringa COMMAND$.
CONST unMese% = 2            ' L'utente richiede un mese specifico.
CONST meseAtt% = 3           ' La richiesta è per un mese di quest'anno.
CONST annoInt% = 4           ' La richiesta è per un intero anno.
CONST stampaInf% = 5         ' Valore aggiuntivo da stampare.
CONST falso% = 0             ' Valore logico FALSE.
CONST vero% = NOT falso%     ' Valore logico TRUE.

Output

END
```

```
FUNCTION NumData& (mese%, giorno%, anno%)

'  La funzione NumData& ritorna un intero lungo che
'     rappresenta la data specificata negli argomenti. Il
'     valore pari a 1 rappresenta: 1 Gennaio 1900.

   annoIniz% = 1900
   Genn% = 1
   GiorPerAn& = 365

'  Se la funzione riceve una data non valida, il valore di
'     ritorno è zero.

   tropPr% = (anno% < annoIniz%)
   meseErr% = (mese% < 1 OR mese% > 12)
   giorErr% = (gio% < 1 OR gio% > GiorniMese%(mese%, anno%))
   IF tropPr% OR meseErr% OR giorErr% THEN
      NumData& = 0
      EXIT FUNCTION
   END IF

'  Diversamente, calcolare i giorni trascorsi dal 1900.

   num& = giorPerAn& * (anno% - annoIniz%)

'  Aggiungere i giorni extra per gli anni bisestili.

   FOR annoCorr% = annoIniz% TO anno% - 1 STEP 4
      IF AnnoBisest%(annoCorr%) THEN num& = num& + 1
   NEXT annoCorr%

'  Contare i giorni dei mesi — fino a
'  escludere il mese specificato.

   FOR MeseCorr% = Genn% TO mese% - 1
      num& = num& + GiorniMese%(meseCorr%, anno%)
   NEXT meseCorr%

'  Aggiungere i giorni fino alla data specificata e
'  ritornare il valore ottenuto.

   num& = num& + giorno%
   NumData& = num&

END FUNCTION

FUNCTION NumGior% (mese%, giorno%, anno%)

'  La funzione NumGior% ritorna un valore da 1 a 7,
'     che rappresenta il giorno della settimana (da
```

```
'     domenica a sabato) della data specificata negli
'     argomenti. In un valore di ritorno pari a zero
'     risulta una data non valida.

   g& = numData&(mese%, giorno%, anno%)
   IF g& <> 0 THEN gds% = g& MOD 7 + 1 ELSE gds% = 0

   NumGior% = gds%

END FUNCTION

SUB Output

'  La procedura Output seleziona una delle molte scelte di
'  output, secondo le istruzioni della linea di
'  comando indicate dall'utente.

   LeggiLiCom operaz%, me%, an%

   IF operaz% >= stampaInf% THEN
      Output% = vero%
      operaz% = operaz% - stampaInf%
      PrepStamp
   ELSE
      Output% = falso%
   END IF
   PRINT
   SELECT CASE operaz%

'  Visualizza il mese corrente ed il successivo
'  se l'utente non ha dato nessuna istruzione o se non
'  sono valide.

      CASE IS <= nessCom%
         meseCorr% = VAL(LEFT$(DATE$, 2))
         annoCorr% = VAL(RIGHT$(DATE$, 4))
         StampOVisual Output%, meseCorr%, annoCorr%
         IF meseCorr% = 12 THEN
            meseCorr% = 1
            annoCorr% = annoCorr% + 1
         ELSE
            meseCorr% = meseCorr% + 1
         END IF
         StampOVisual Output%, meseCorr%, annoCorr%
'  Visualizza un solo mese.
      CASE unMese%, meseAtt%
         IF an% = THEN anno% = VAL(RIGHT$(DATE$, 4)) ELSE anno% = an%
         StampOVisual Output%, me%, anno%

'  Visualizza tutti i dodici mesi dell'anno specificato.
```

```
      CASE annoInt%
         FOR mese% = 1 TO 12
            StampOVisual Output%, mese%, an%
         NEXT mese%

   END SELECT

END SUB

FUNCTION AnnoBisest% (anno%)

'  La funzione AnnoBisest% ritorna un valore booleano che
'  indica se un determinato anno è bisestile.

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)

   AnnoBisest% = divPer4% AND (secolo% IMP secolo400%)

END FUNCTION

FUNCTION GiorniMese% (numMese%, anno%)

'  La funzione GiorniMese% ritorna un intero da 28 a 31,
'     che indica il numero di giorni in un determinato
'     mese. L'argomento anno% viene richiesto per
'     determinare il numero dei giorni di febbraio.

'  Ritornare un valore pari a zero per un mese non valido.

   IF numMese% < 1 OR numMese% > 12 THEN
      GiorniMese% = 0
      EXIT FUNCTION
   END IF

'  Diversamente, determinare il numero dei giorni nel mese.

   SELECT CASE numMese%

'  Febb.

      CASE 2
         giorni% = 28
         IF AnnoBisest%(anno%) THEN giorni% = giorni% + 1
'  Apr, Giu, Set e Nov.

      CASE 4, 6, 9, 11
         giorni% = 30
'  I mesi restanti.
```

```
      CASE ELSE
         giorni% = 31
   END SELECT

   GiorniMese% = giorni%

END FUNCTION

FUNCTION Mese$ (numMese%)

'  La funzione Mese$ prende un argomento da 1 a 12 e
'     ritorna il nome del mese corrispondente. Dato un
'     numero non valido di mese, la funzione ritorna una
'     stringa vuota.

        m$ = "Gen Feb Mar Apr "
   m$ = m$ + "Mag Giu Lug Ago "
   m$ = m$ + "Set Ott Nov Dic "

   IF numMese% >= 1 AND numMese% <= 12 THEN
      strMe$ = MID$(m$, (numMe% - 1) * 10 + 1, 10)
      strMe$ = RTRIM$(strMe$)
   ELSE
      strMe$ = ""
   END IF
   Mese$ = strMe$

END FUNCTION

PrepStamp:

'  Il sottoprogramma PrepStamp dà all'utente la
'     possibilità di preparare la stampante.

   PRINT "Premere la barra spaziatrice quando la stampante è pronta. ";
   barSp$ = ""
   DO WHILE barSp$ <> " "
      barSp$ = INKEY$
   LOOP
   PRINT

END SUB

SUB StampMese (mese%, anno%)

'  La procedura StampMese stampa il mese specificato negli
'     argomenti.

'  Determinare il mese, il giorno d'inizio ed il numero di giorni.
   primGior% = NumGior%(mese%, 1, anno%)
```

```
   lunghMese% = GiorniMese%(mese%, anno%)
   strMe$ = Mese$(mese%)

'  Stampare il nome del mese e le abbreviazioni del giorno.

   LPRINT SPACE$ (12 - LEN(strMe$) \ 2); strMe$; anno%
   LPRINT
   LPRINT "D  L  M  M  G  V  S"
   LPRINT " "; STRING$(26, "-")

'  Stampare i giorni in colonna.

   dataCorr% = 0
   DO
      FOR giorCorr% = 1 TO 7
         PrimGior% = (dataCorr% = primGior% AND dataCorr% = 0)
         NelMese% = (dataCorr% > 0 AND dataCorr% < lunghMese%)
         IF PrimGior% OR NelMese% THEN
            dataCorr% = dataCorr% + 1
            LPRINT USING " ##"; dataCorr%;
         ELSE
            PRINT "  ";
         END IF
      NEXT giornCorr%
      LPRINT
   LOOP UNTIL dataCorr% = lunghMese%

END SUB

SUB StampOVisual (st%, me%, an%)

'  La procedura StampOVisual chiama o la routine
'     StampMese per stampare un mese, o la routine
'     VisualMese per visualizzare un mese sullo schermo.
'     La scelta tra queste due opzioni dipende dal valore
'     logico passato a st%.

   IF st% THEN

'  Stampare il mese.

      StampMese me%, an%
      LPRINT

ELSE
'  Visualizzare il mese sullo schermo.
      VisualMese me%, an%
      PRINT
   END IF
```

```
END SUB

SUB LeggiLinCom (qualOutput%, mese%, anno%)

'  La procedura LeggiLinCom legge la linea linCom$ che
'  l'utente fornisce dal prompt del DOS.

'  Eliminare gli spazi terminali dalla fine di COMMAND$.

   linCom$ = RTRIM$(COMMAND$)

'  Se l'utente non dà nessun parametro, la stringa
'  linCom$ è vuota.

   IF LEN(linCom$) = 0 THEN
      qualOutput% = nessCom%

'  Diversamente, iniziare a leggere la stringa.
   ELSE
      posVirg% = INSTR(linCom$, ",")

'  Se la stringa contiene una virgola, separare i due elementi.

      IF posVirg% <> 0 THEN
         mese% = VAL(LEFT$(linCom$, posVirg%))
         anno% = VAL(MID$(linCom$, posVirg% + 1))
         possMese% = (mese% >= 1 AND mese% <= 12)
         possAnno% = (anno% >= 1900 AND anno% <= 2500)
         IF possMese% AND anno% THEN
            qualOutput% = unMese%
         ELSE
            qualOutput% = comandErr%
         END IF

'  Diversamente, leggere il singolo parametro come un anno o un mese.

      ELSE
         inVal% = VAL(Lincom$)
         IF inVal% >= 1 AND inVal% <= 12 THEN
            qualOutput% = meseAtt%
            mese% = inVal%
            anno% = 0
         ELSEIF inVal% >= 1900 AND inVal% <= 2500 THEN
            qualOutput% = annoCompl%
            anno% = inVal%
            mese% = 0
            ELSE
               qualOutput% = comErr%
               anno% = 0
               mese% = 0
```

```
         END IF
      END IF
   END IF

' Stabilisce se si debba stampare l'output.

   IF RIGHT$(Lincom$, 2) = "/P" THEN
      qualOutput% = qualOutput% + infoStamp%
   END IF

END SUB

SUB VisualMese (mese%, anno%)

' La procedura VisualMese visualizza il mese del
'    calendario specificato negli argomenti.

' Determinare il mese, il giorno d'inizio ed il numero di
' giorni.

   primGior% = NumGior%(mese%, 1, anno%)
   lunghMese% = GiorniMese%(mese%, anno%)
   strMe$ = Mese$(mese%)

' Visualizzare il nome del mese e le abbreviazioni del
' giorno.
   PRINT SPACE$(12 - LEN(strMe$) \ 2); strMe$; anno%
   PRINT
   PRINT "  D  L  M  M  G  V  S"
   PRINT " "; STRING$(26, "-")

' Visualizzare i giorni in colonna.
   dataCorr% = 0
   DO
      FOR giorCorr% = 1 TO 7
         PrimGior% = (dataCorr% = primGior% AND dataCorr% = 0)
         NelMese% = (dataCorr% > 0 AND dataCorr% < lunghMese%)
         IF PrimGior% OR NelMese% THEN
            dataCorr% = dataCorr% + 1
            PRINT USING " ##"; dataCorr%;
         ELSE
            PRINT "   ";
         END IF
      NEXT giorCorr%
      PRINT
   LOOP UNTIL dataCorr% = lunghMese%

END SUB
```

Capitolo 30

Il programma Agenda

Il programma *Agenda* organizza un calendario degli appuntamenti giornalieri in un archivio su disco. Si può usare questo semplice programma per segnare i propri appuntamenti e gli impegni di lavoro. Il programma visualizza sullo schermo il programma del giorno con i vari orari e consente di modificare gli appuntamenti esistenti o di aggiungerne di nuovi. Inoltre, *Agenda* illustra varie tecniche di input ed output che coinvolgono la tastiera, lo schermo e l'utilizzo di file di dati su disco. *Agenda* utilizza due diversi file su disco: uno ad accesso casuale che memorizza il calendario ed uno ad accesso sequenziale che serve da indice del database, così che il programma è in grado di localizzare velocemente i singoli record dei giorni. Nella sua forma corrente, *Agenda* è progettata per essere compilata verso un file EXE e poi attivata direttamente dal prompt del DOS. Il programma esempio *GestioneDate* nel Capitolo 31 realizza una buona interfaccia utente per il programma *Agenda*. *GestioneDate* visualizza il calendario di un mese ed offre all'utente una semplice serie di tecniche da tastiera per selezionare un giorno. Quando l'utente ha fatto una scelta, *GestioneDate* invia la data in questione al programma *Agenda* che, di volta in volta, visualizza sullo schermo i vari appuntamenti. *GestioneDate* e *Agenda* devono rimanere sul disco come programmi compilati separati; in questo modo si può sempre usare *Agenda* senza passare attraverso l'interfaccia *GestioneDate*. Questa parte descrive *Agenda* come programma indipendente: si possono trovare informazioni complete sull'utilizzo di *GestioneDate* nel capitolo seguente.

ATTIVAZIONE DEL PROGRAMMA

Agenda consente di memorizzare gli appuntamenti per una data scelta dall'utente. Ogni volta che si attiva il programma, esso visualizza sullo schermo gli appuntamenti del giorno selezionato. Il programma accetta la data dal prompt del DOS, nel formato mm/gg/aa; ad esempio si può scrivere:

```
C>AGENDA 11/26/90
```

In risposta, il programma cerca nel database del calendario se c'è già una serie memorizzata di impegni per la data in questione. Nel caso in cui la data sia memorizzata, il programma visualizza sullo schermo gli appuntamenti del giorno.

```
Lun., Nov. 26, 1990
---------------

  7:00 a.m. >
  8:00 a.m. >
  9:00 a.m. > Incontro con M.C. nella sala dei congressi.
 10:00 a.m. > Telefonare a Franca Scala. (02) 7235876
 11:00 a.m. >
 12:00      > Pranzo con Bassi. Rist. Italia, Via Larga, 15.
  1:00 p.m. >
  2:00 p.m. >
  3:00 p.m. >
  4:00 p.m. > Preparat. per viaggio a Roma (sett. pross.)
  5:00 p.m. >
  6:00 p.m. >
```

Se, però, non è memorizzato nessun appuntamento per il giorno selezionato, sullo schermo appare l'agenda vuota:

```
Mar., Nov. 27, 1990
---------------

  7:00 a.m. >
  8:00 a.m. >
  9:00 a.m. >
 10:00 a.m. >
 11:00 a.m. >
 12:00      >
  1:00 p.m. >
  2:00 p.m. >
```

```
3:00 p.m. >
4:00 p.m. >
5:00 p.m. >
6:00 p.m. >
```

Come si può vedere, vi sono linee disponibili per gli appuntamenti in corrispondenza delle varie ore del giorno, dalle 7:00 del mattino fino a mezzogiorno e da 1:00 alle 6 del pomeriggio. Sotto queste linee il programma visualizza il seguente prompt d'input:

```
Immettere l'ora (dalle <7> alle <12> o dalla <1> alle <6>)
per un nuovo appuntamento oppure premere <Q> per uscire: -
```

Per inserire un nuovo appuntamento nel giorno (o cambiarne uno) bisogna iniziare immettendo l'ora, un intero da 7 a 12 o da 1 a 6. In risposta il programma visualizza un altro prompt nella parte inferiore dello schermo, dando così l'opportunità di inserire la descrizione in una linea; ad esempio:

```
10:00 a.m. appuntamento: -
```

Si possono immettere fino a 50 caratteri di testo per descrivere l'appuntamento. Quando si preme il tasto Invio, la nuova immissione appare sullo schermo vicino all'ora selezionata e il programma richiede di scegliere un'altra ora:

```
Immettere l'ora (dalle <7> alle <12> o dalla <1> alle <6>)
per un nuovo appuntamento oppure premere <Q> per uscire: -
```

Si possono inserire nuovi appuntamenti o modificare quelli esistenti per ogni sequenza di ore. Quando si ha terminato di lavorare con l'agenda del giorno, bisogna premere Q come risposta al prompt. Il programma, poi, visualizza sullo schermo un altro prompt che chiede la conferma per salvare su disco gli appuntamenti del giorno nell'archivio agenda:

```
Salvi gli appuntamenti del giorno? <S> o <N> -
```

Se si risponde premendo il tasto S, il programma salva il record come lo si vede sullo schermo. Se si preme il tasto N, il programma tralascia le nuove immissioni o le modifiche fatte. In ogni caso, questa operazione completa l'esecuzione corrente del programma. Se si desidera esaminare o modificare

l'agenda di un altro giorno, bisogna riattivare il programma *Agenda*. (Uno dei vantaggi del programma *GestioneDate*, presentato nel Capitolo 31, è l'opportunità di richiamare tutte le date desiderate durante una data attivazione del programma.) Comunque, si può attivare il programma dal prompt del DOS in vari modi. In primo luogo, se si inserisce solo il nome del programma, senza specificare nessuna data, *Agenda* crede che si voglia lavorare con la data odierna:

```
C>AGENDA
```

In alternativa, si può inserire la data nel formato mm/gg, che specifica una data dell'anno in corso. Infine, se capita che si invii al programma una data non valida o una stringa che il programma non riesce a leggere come una data, sullo schermo appare il seguente messaggio d'aiuto:

```
------------------------------------------
Data non valida.

Il programma Agenda si aspetta una data
nel formato mm/gg/aa oppure mm/gg/aaaa.
(Si presuppone che un anno a due cifre
sia del ventesimo secolo.) Si può inserire
la data anche come mm/gg, e in questo caso
il programma usa l'anno corrente per
completare la data.
------------------------------------------
```

Se si dimentica come il programma funziona si può sempre ottenere il messaggio precedente attivando il programma nel seguente modo:

```
C>AGENDA HELP
```

Non c'è uno specifico limite per il numero di date che si possono immettere nel database del calendario; il programma mantiene due file separati:

- AGENDA.DAT è il file ad accesso casuale che memorizza le immissioni del giorno.
- AGENDA.NDX è un file sequenziale che serve da indice del database.

La prima volta che si attiva il programma *Agenda* e si fissano degli appuntamenti nell'archivio, entrambi questi file sono creati su disco. Per le

esecuzioni successive, il programma dev'essere in grado di localizzare ed aprire entrambi i file. Se il file AGENDA.DAT esiste, mentre AGENDA.NDX è stato inavvertitamente distrutto, il programma visualizza questo messaggio:

```
Errore del file:
----------------

AGENDA.DAT (il file dell'archivio dell'agenda)
esiste, mentre il file indice, AGENDA.NDX,
non può essere trovato. Il programma Agenda non
può continuare.

Premere un tasto qualsiasi per terminare il programma.
```

In questo caso si potrebbe voler salvare il corrente AGENDA.DAT con un nome di backup (ad esempio AGENDA.BAK) e poi riattivare il programma. *Agenda* inizierà creando l'archivio del calendario e poi ricreando due nuovi file su disco.

LA STRUTTURA DEL PROGRAMMA

Il programma *Agenda* dimostra le tecniche specifiche di gestione del database in QuickBASIC. Il programma contiene ampi commenti che aiutano a capire particolari procedure e passaggi. Inoltre, il programma usa un certo numero di procedure per la manipolazione delle date che sono già state sviluppate e descritte per il programma *Mese*, presentato nel Capitolo 29. All'inizio del listato del programma *Agenda*, si possono vedere varie strutture di dati che il programma crea per la gestione del database del calendario. Il tipo definito dall'utente *recGior* è una struttura record usata per definire ed aprire il file AGENDA.DAT e per eseguire le operazioni di input e output con il file. Molte procedure nel programma dichiarano delle variabili record che appartengono a questo tipo. Una volta che un record è stato letto in memoria dal file, il programma copia le informazioni del record in due vettori chiamati *matt* e *pomer*. Insieme, questi due vettori forniscono una struttura vantaggiosa per la gestione dei dati del programma. Appena prima che il programma scriva un nuovo record nel file, i contenuti dei vettori *matt* e *pomer* vengono ricopiati in una variabile record di tipo *recGior*. Questa variabile diventa poi il mezzo per scrivere il record nel file. Il programma dichiara anche un altro tipo di

struttura chiamata *immInd* e crea un vettore di record di tipo *immInd* chiamato *indData*. Le informazioni provenienti dal file AGENDA.NDX vengono poi copiate in questo vettore globale. Quando tutte le strutture dati necessarie sono state inizializzate, il programma esegue una chiamata alla procedura *AprDatab* per aprire il database del calendario. Questa procedura apre AGENDA.DAT come file ad accesso casuale e determina il numero di record che sono correntemente memorizzati nel file. Se il numero è maggiore di zero, il programma presuppone che il file indice, AGENDA.NDX, sarà anch'esso disponibile sul disco. *AprDatab* esegue una chiamata alla procedura *AprIndDatab* per aprire questo file sequenziale di dati e ne legge i contenuti nel vettore *indData*. Successivamente il programma chiama la procedura *LeggiInData* per leggere la linea COMMAND$, cioè l'argomento che l'utente ha fornito al programma *Agenda* dal prompt del DOS. Presupponendo che l'utente abbia inserito una data valida, una chiamata alla procedura *CreaAgenda* fa iniziare il ciclo principale del programma. Ecco i compiti che il programma esegue:

- Cerca la data precisata nell'indice del database. (Una chiamata alla funzione *CercData%* esegue la ricerca.)
- Se la data viene trovata nell'indice, copia il record corrispondente dal file del database nei vettori *matt* e *pomer*. (La procedura *LeggiGior* legge il record dal database.)
- Visualizza sullo schermo l'agenda del giorno. (La procedura *VisualMese* visualizza un'agenda vuota, se la data dell'utente non viene trovata nel database oppure un'agenda completa con tutti gli appuntamenti segnati, se esiste già un record per la data in questione.)
- Gestisce il dialogo d'input per inserire i nuovi appuntamenti. (La procedura *Corr* legge i record dal database.)
- Se l'utente ha inserito degli appuntamenti per un nuovo record dell'archivio, viene ricreato l'indice e memorizzato nel file sequenziale AGENDA.NDX. (La procedura *IndCorr* esegue l'aggiornamento dell'indice. Prima che l'indice sia salvato su disco, il programma chiama *OrdIndData* per riordinarlo dopo l'aggiunta di un nuovo record.)
- Scrive il nuovo record nel file del database. (La procedura *SalvGior* inizia copiando il record dai vettori *matt* e *pomer* nei singoli campi del record *giorno* e poi scrive *giorno* come un record nel file del database.)

Molte di queste operazioni, naturalmente, non vengono viste dall'utente, ma tutte insieme offrono una tecnica sicura per gestire l'archivio del calendario.

IL LISTATO DEL PROGRAMMA AGENDA

```
' AGENDA.BAS
' Mantiene su disco un database per memorizzare l'agenda
' personale dell'utente.

' Il programma Agenda legge una data opzionale dalla
' linea di comando del DOS. La data può presentarsi nella
' forma mm/gg/aa o mm/gg/aaaa. (Se si indica un anno con
' due cifre, il programma dà per scontato che si tratti
' di un anno del ventesimo secolo.) Come risposta a
' questo input dalla linea di comando, il programma cerca
' nel database dell'agenda un record che corrisponda alla
' data specificata. Se un record esiste, il programma lo
' visualizza sullo schermo e dà l'opportunità di vederlo
' e/o di modificare gli appuntamenti del giorno. In caso
' contrario, il programma visualizza un foglio bianco per
' la data specificata e consente di iniziare ad immettere
' delle informazioni. Il record nuovo o
' modificato viene poi memorizzato nell'archivio
' dell'agenda.

' Se non si inserisce una data dalla linea di comando, il
' programma visualizza il record — esistente o
' vuoto — del giorno.

' Il programma tiene due file sul disco: AGENDA.DAT è il
' database e AGENDA.NDX è un file indice che
' consente al programma di localizzare nell'archivio un
' singolo record.

DECLARE SUB CancLinea (Numlinea%)
DECLARE SUB Corr (salvCorr%)
DECLARE SUB VisualGior ()
DECLARE SUB VisualAiu ()
DECLARE SUB EsegAgend (mese%, giorno%, anno%)
DECLARE SUB AprDatab ()
DECLARE SUB AprIndData ()
DECLARE SUB LeggiInfCal ()
DECLARE SUB LeggiGior (numRec%)
DECLARE SUB LeggiInData (mese%, giorno%, anno%)
DECLARE SUB IndCorr (dataCorr&)
DECLARE SUB SalvGior (numRec%)
DECLARE SUB OrdIndData ()
```

```
DECLARE FUNCTION NumGior% (mese%, giorno%, anno%)
DECLARE FUNCTION NumData& (mese%, giorno%, anno%)
DECLARE FUNCTION DataCompl$ (mese%, giorno%, anno%)
DECLARE FUNCTION AnnoBisest% (anno%)
DECLARE FUNCTION GiorniMese% (numMese%, anno%)
DECLARE FUNCTION CercData% (giorFiss&)

CONST FALSE = 0            ' Il valore logico FALSE.
CONST TRUE = NOT FALSE     ' Il valore logico TRUE.
CONST lunghElem = 50       ' Lunghezza massima di
                           ' un'immissione nell'agenda.
CONST fileNoTrov = 53      ' Numero di errore per "File non trovato".

'  Definisce il tipo record per un giorno dell'agenda.

TYPE recGior
   matt7 AS STRING * lunghElem
   matt8 AS STRING * lunghElem
   matt9 AS STRING * lunghElem
   matt10 AS STRING * lunghElem
   matt11 AS STRING * lunghElem
   matt12 AS STRING * lunghElem
   pomer1 AS STRING * lunghElem
   pomer2 AS STRING * lunghElem
   pomer3 AS STRING * lunghElem
   pomer4 AS STRING * lunghElem
   pomer5 AS STRING * lunghElem
   pomer6 AS STRING * lunghElem
END TYPE

'  Definisce il tipo del record del file AGENDA.NDX.

TYPE ImmInd
   dataLung AS LONG
   posData AS INTEGER
END TYPE

'  La variabile intera globale contImm% rappresenta il numero
'  delle immissioni attualmente memorizzate nel database
'  AGENDA.DAT. Il vettore globale indData memorizza
'  l'intero indice durante l'esecuzione del programma.

COMMON SHARED contImm%, indData() AS immInd

'  Definisce i due vettori globali, matt e pomer, progettati
'  per memorizzare le informazioni relative ad un giorno
'  durante l'esecuzione del programma.

DIM SHARED matt(7 TO 12) AS STRING * lunghElem
DIM SHARED pomer(1 TO 6) AS STRING * lunghElem
```

```
' Crea i vettori per memorizzare le informazioni del calendario.

DIM SHARED Mese$(12), giorMese%(12), giorSett$(7)
LeggiInfCal

AprDatab                    ' Apre il database dell'agenda.

LeggiInData me%, gio%, an%  ' Legge la data dalla linea di comando.

' Se la data della linea di comando è valida,
' visualizza il foglio dell'agenda corrispondente;
' altrimenti compare una videata d'aiuto.

IF an% <> 0 THEN
   CLS
   EsegAgenda me%, gio%, an%
ELSE
   VisualAiu
END IF

END

' Informazioni del calendario:

DATA Gen, 31, Feb, 29, Mar, 31, Apr, 30, Mag, 31, Giu, 30
DATA Lug, 31, Ago, 31, Set, 30, Ott, 31, Nov, 30, Dic, 31
DATA Dom, Lun, Mar, Mer, Gio, Ven, Sab

' Routine di errore. Assume il controllo se il file
' indice non si trova sul disco.

noFileInd:
   IF ERR = FileNoTrov THEN
      CLS
      PRINT "Errore del file:"
      PRINT "--------"
      PRINT "AGENDA.DAT (il file del database agenda) esiste,"
      PRINT "ma il file dell'indice, AGENDA.NDX, non si trova."
      PRINT "Il programma Agenda non può continuare."
      PRINT
      PRINT "Premere un tasto per terminare il programma."
      SLEEP
      END
   ELSE
      ON ERROR GOTO 0
   END IF

SUB CancLinea (numLinea%)
```

```
'  Il sottoprogramma CancLinea cancella una linea
'     specifica dal testo posto sullo schermo.

   VIEW PRINT numLinea% TO numLinea%
   CLS
   VIEW PRINT

END SUB

FUNCTION NumData& (mese%, giorno%, anno%)

'  La funzione NumData& ritorna un intero lungo che rappresenta
'     la data specificata negli argomenti. Il valore
'     pari a 1 rappresenta: 1 Gennaio 1900.

   annoIniz% = 1900
   Genn% = 1
   GiorPerAn& = 365

'  Se la funzione riceve una data non valida, il valore di
'  ritorno è zero.

   tropPr% = (anno% < annoIniz%)
   meseErr% = (mese% < 1 OR mese% > 12)
   giorErr% = (gio% < 1 OR gio% > GiorniMese%(mese%, anno%))
   IF tropPr% OR meseErr% OR giorErr% THEN
      NumData& = 0
      EXIT FUNCTION
   END IF

'  Altrimenti, calcola i giorni trascorsi dal 1900.

   num& = giorPerAn& * (anno% - annoIniz%)

'  Aggiunge i giorni extra per gli anni bisestili.

   FOR annoCorr% = annoIniz% TO anno% - 1 STEP 4
      IF AnnoBisest%(annoCorr%) THEN num& = num& + 1
   NEXT annoCorr%

'  Conta i giorni dei mesi fino al mese specificato escluso.

   FOR MeseCorr% = Genn% TO mese% - 1
      num& = num& + giorniMese%(meseCorr%, anno%)
   NEXT meseCorr%

'  Aggiunge i giorni fino alla data specificata e
'  ritorna il valore ottenuto.
```

```
   num& = num& + gior%
   NumData& = num&

END FUNCTION

FUNCTION NumGior% (mese%, giorno%, anno%)

'  La funzione NumGior% ritorna un valore da 1 a 7,
'     che rappresenta il giorno della settimana (da
'     domenica a sabato) della data specificata negli
'     argomenti. Una data non valida risulta in un valore di
'     ritorno pari a zero.

   g& = numData&(mese%, giorno%, anno%)
   IF g& <> 0 THEN gds% = g& MOD 7 + 1 ELSE gds% = 0

   NumGior% = gds%

END FUNCTION

SUB VisualGior

'  Il sottoprogramma VisualGior visualizza sullo schermo
'     l'agenda di un giorno selezionato. Se il record
'     relativo è stato trovato in AGENDA.DAT, le
'     immissioni vengono memorizzate nei
'     vettori globali matt e pomer. Altrimenti,
'     VisualGior visualizza un foglio bianco dell'agenda.

'  Visualizza le ore del mattino e il mezzogiorno.

   FOR ora% = LBOUND(matt) TO UBOUND(matt) - 1
      PRINT USING "##:00 a.m. > "; ora%;
      PRINT matt(ora%)
   NEXT ora%

   PRINT "12:00 mezz > "; matt(12)

'  Visualizza le ore del pomeriggio.

   FOR ora% = LBOUND(pomer) TO UBOUND(pomer)
      PRINT USING " #:00 p.m. > "; ora%;
      PRINT pomer(ora%)
   NEXT ora%

END SUB

SUB VisualAiu

'  Il sottoprogramma VisualAiu viene chiamato se l'utente
'     inserisce una data non valida dalla linea di comando.
```

```
   PRINT
   PRINT STRING$(37, "-")
   IF INSTR(COMMAND$, "AIUTO") = 0 THEN
      PRINT "Data non valida."
      PRINT
   END IF
   PRINT "Il programma Agenda si aspetta una data"
   PRINT "nel formato mm/gg/aa oppure mm/gg/aaaa."
   PRINT "(Si presuppone che un anno a due cifre sia"
   PRINT "del ventesimo secolo.) Si può"
   PRINT "inserire la data anche come mm/gg, e in"
   PRINT "questo caso il programma fornisce"
   PRINT "l'anno corrente per completare la data."
   PRINT "Se non si inserisce nessuna data nella"
   PRINT "linea di comando, il programma Agenda"
   PRINT "utilizza la data corrente."
   PRINT STRING$(37, "-")

END SUB

SUB EsegAgenda (mese%, giorno%, anno%)

'  Il sottoprogramma EsegAgenda controlla la
'     visualizzazione dell'agenda di un giorno e il
'     processo con il quale vengono ricevute le nuove
'     informazioni date dall'utente a tastiera

'  Inizia cercando la data specificata nel database di
'  Agenda.

   dataFiss& = NumData&(mese%, giorno%, anno%)
   possTrov% = CercData%(dataFiss&)

'  Se il record esiste, viene letto dal database nei due
'  vettori globali chiamati matt e pomer.

   IF posTrov% <> 0 THEN LeggiGior posTrov%

'  Visualizza la data nella parte superiore dello schermo.

   strData$ = DataCompl$(mese%, giorno%, anno%)
   PRINT strData$
   PRINT STRING$(LEN(strData$), "-")
   PRINT

'  Visualizza l'agenda del giorno dal database Agenda o
'  un foglio bianco per accettare nuove informazioni.

   VisualGior
```

```
' Chiamare il sottoprogramma Corr per instaurare un
' dialogo di input per le nuove informazioni dell'agenda.

Corr OkPerSalv%

' Se l'utente conferma (OkPerSalv% è TRUE), salva il
' record nuovo (o corretto) sul disco nel database Agenda.

IF OkPerSalv% THEN

' Se questo è un nuovo record, aumentare di 1 il valore
' di contImm% e chiamare IndCorr per ricreare l'indice di
' Agenda.

      IF posTrov% = 0 THEN
         contImm% = contImm% + 1
         IndCorr dataFiss&
         possTrov% = contImm%
      END IF
      SalvGior posTrov%
   END IF
END SUB

FUNCTION DataCompl$ (mese%, giorno%, anno%)

' Dati tre interi che rappresentano una data, la funzione
'    DataCompl$ ritorna una stringa data; ad esempio,
'    Lun., Nov. 26, 1990. Per eseguire il proprio
'    compito, questa funzione esegue una chiamata alla
'    funzione NumGior% e seleziona i dati dai vettori
'    globali giorSett$ e nomiMese$.

   gio% = NumGior%(mese%, giorno%, anno%)
   strData$ = giorSett$(gio%) + ".," + nomiMese$(mese%)

' I nomi di tutti i mesi vengono abbreviati.

   IF mese% <> 5 THEN strData$ = strData$ + "."
   strData$ = strData$ + STR$(giorno%) + "," + STR$(anno%)

   DataCompl$ = strData$

END FUNCTION

SUB Corr (salvCorr%)

' Il sottoprogramma Corr instaura un dialogo di input per
'    le nuove immissioni dell'agenda per la data
'    selezionata. Nella variabile parametro salvCorr%, la
'    routine ritorna un valore logico che indica se
'    l'utente vuole salvare le correzioni.
```

```
   salvCorr% = FALSE

'  Continuare il dialogo finché l'utente non preme il
'  tasto Q.

   eseg% = FALSE
   DO
      VIEW PRINT 19 TO 23
      CLS
      PRINT "Immettere l'ora (dalle <7> alle <12> o dalla <1>
                                 alle <6>) "
      INPUT "per un nuovo appuntamento, oppure immettere <Q>
                                 per uscire: ", InVal$
      PRINT
      IF INSTR(UCASE$(inVal$), "Q") <> 0 THEN
         eseg% = TRUE
      ELSE
         inVal% = VAL(inVal$)
         IF inVal% >= 1 AND inVal% <= 12 THEN salvCorr% = TRUE

'  Accetta un'immissione per un appuntamento del giorno.
'  (Memorizza l'immissione nel vettore chiamato matt.)

      SELECT CASE inVal%
         CASE 7 TO 11
            PRINT USING "##:00 a.m. "; inVal%;
            LINE INPUT "appunt: "; matt(inVal%)
            CancLinea inVal% - 3
            VIEW PRINT inVal% - 3 TO inVal% - 2
            PRINT USING "##:00 a.m. > "; inVal%;
            PRINT matt(inVal%)

'  Accetta l'immissione per un appuntamento a
'  mezzogiorno.

         CASE 12
            LINE INPUT "12:00 appunt mezz: "; matt(12)
            CancLinea 9
            VIEW PRINT 9 TO 10
            PRINT "12:00 mezz > "; matt(12)

'  Accetta un'immissione per un appuntamento nel pomeriggio.
'  (Memorizzare l'immissione nel vettore chiamato pomer.)

         CASE 1 TO 6
            PRINT USING "#:00 p.m. "; inVal%;
            LINE INPUT "appunt: "; pomer(inVal%)
            CancLinea inVal% + 9
            VIEW PRINT inVal% + 9 TO inVal% + 10
            PRINT USING "#:00 p.m. > "; inVal%;
```

```
          PRINT pomer(inVal%)
      END SELECT

   END IF
LOOP UNTIL eseg%
VIEW PRINT
LOCATE 23, 5, 1

'  Chiede la conferma dell'utente prima di salvare questo
'  record corretto nel database.

   IF salvCorr% THEN
      PRINT "Salvo gli appuntamenti del giorno? <S> o <N> ";
      DO
         risp$ = UCASE$(INKEY$)
      LOOP UNTIL LEN(risp$) > 0 AND INSTR("SN", risp$) <> 0
      PRINT risp$;
      salvCorr% = (risp$ = "S")
   END IF
END SUB

FUNCTION AnnoBisest% (anno%)

' La funzione AnnoBisest% ritorna un valore booleano che
' indica se un determinato anno è bisestile.

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)

   AnnoBisest% = divPer4% AND (secolo% IMP secolo400%)

END FUNCTION

FUNCTION GiorniMese% (numMese%, anno%)

'  La funzione GiorniMese% ritorna un intero da 28 a 31,
'     che indica il numero di giorni in un determinato
'     mese. L'argomento anno% viene richiesto per
'     determinare il numero dei giorni di febbraio.

'  Ritorna un valore pari a zero per un mese non valido.

   IF numMese% < 1 OR numMese% > 12 THEN
      GiorniMese% = 0
      EXIT FUNCTION
   END IF

'  Diversamente, determina il numero dei giorni nel mese.
```

```
   giorni% = giorMese%(numMese%)
   IF numMese% = 2 AND NOT AnnoBisest%(anno%) THEN giorni% = giorni% - 1

   GiorniMese% = giorni%

END FUNCTION

SUB AprDatab

'  Il sottoprogramma AprDatab apre il database Agenda (se
'     esiste già) e determina il numero di record che il
'     database attualmente contiene.

'  Definisce una variabile record temporanea per
'  determinare le dimensioni del record del
'  database.

   DIM giorno AS recGior

'  Apre il database nel modo ad accesso casuale.

   OPEN "AGENDA.DAT" FOR RANDOM AS #1 LEN = LEN(giorno)

'  Utilizza la variabile record per calcolare il numero attuale
'  di record nel database. Memorizza il conteggio nella variabile
'  globale chiamata contImm%.

   contImm% = LOF(1) / LEN(giorno)

'  Se il database contiene dei record, chiama il sottoprogramma
'  AprIndData per leggere il file indice.

   IF contImm% <> 0 THEN AprIndData

END SUB

SUB AprIndData

'  Il sottoprogramma AprIndData legge il file indice per
'     il database Agenda e memorizza l'indice nel vettore
'     globale chiamato indData.

'  Inizia fissando le dimensioni richieste del vettore
'     indData, basato sul conteggio corrente del record del
'     database. (La struttura indData è un vettore dinamico
'     dei record tipo immInd.)

   REDIM indData(immCont%) AS immInd

'  Apre il file indice per una lettura sequenziale.
```

```
'  Crea una gestione degli errori per avere il controllo
'  nel caso in cui non si trovi il file indice.

   ON ERROR GOTO nessFileInd
      OPEN "AGENDA.NDX" FOR INPUT AS #2
   ON ERROR GOTO 0

'  Legge l'indice nel vettore indData.

   FOR imm% = 1 TO contImm%
      INPUT #2, indData(imm%).dataLung, indData(imm%).posData
   NEXT imm%

'  Chiude temporaneamente il file indice.

   CLOSE #2

END SUB

SUB LeggiInfCal

'  Il sottoprogramma LeggiInfCal legge delle informazioni
'     da una serie di istruzioni DATA che si trovano
'     all'inizio del programma: i 12 nomi dei mesi ed i
'     corrispondenti numeri dei giorni dei mesi ed i nomi
'     dei 7 giorni. Queste informazioni vengono
'     memorizzate nei tre vettori condivisi chiamati
'     nomiMese$, giorMese% e giorSett$.

   FOR i% = 1 TO 12
      READ nomiMese$(i%)
      READ giorMese%(i%)
   NEXT i%

   FOR i% = 1 TO 7
      READ giorSett$(i%)
   NEXT i%

END SUB

SUB LeggiGior (numRec%)

'  Il sottoprogramma LeggiGior legge un record dal
'     database Agenda e poi memorizza le informazioni
'     dell'agenda nei vettori globali chiamati matt e pomer.

'  Iniziare definendo una variabile record per leggere
'     inizialmente il record per la data fissata.

   DIM giorFiss AS recGior
```

```
' Leggere un record dalla posizione specificata nel database Agenda,
' che è già aperto in modo ad accesso casuale.

   GET #1, numRec%, giorFiss

' Copia i dati del record nei vettori matt e pomer.
' Questi due vettori si rivelano essere il modo più
' vantaggioso per memorizzare i dati per i compiti del
' resto del programma.

   matt(7) = giorFiss.matt7
   matt(8) = giorFiss.matt8
   matt(9) = giorFiss.matt9
   matt(10) = giorFiss.matt10
   matt(11) = giorFiss.matt11
   matt(12) = giorFiss.matt12
   pomer(1) = giorFiss.pomer1
   pomer(2) = giorFiss.pomer2
   pomer(3) = giorFiss.pomer3
   pomer(4) = giorFiss.pomer4
   pomer(5) = giorFiss.pomer5
   pomer(6) = giorFiss.pomer6

END SUB

SUB LeggiInData (mese%, giorno%, anno%)

' Il sottoprogramma LeggiInData legge la stringa data che
'    l'utente fornisce sulla linea di comando del DOS.
'    Questa routine restituisce la data al programma,
'    chiamante per indirizzo, nei parametri mese%,
'    giorno%, anno%.

' I caratteri slash devono essere usati nella data come separatori.

CONST slash = "/"

' Se l'utente non fornisce nessun parametro della linea di comando,
'    si considera quella corrente come data fissata per l'agenda.

   inData$ = RTRIM$(COMMAND$)
   IF LEN(inData$) = 0 THEN
      mese% = VAL(LEFT$(DATE$, 2))
      giorno% = VAL(MID$(DATE$, 4, 2))
      anno% = VAL(RIGHT$(DATE$, 4))
      EXIT SUB
   END IF

' Cerca uno o due slash nella stringa proveniente dalla linea di comando.
```

```
   posSlash1% = INSTR(inData$, slash)
   posSlash2% = INSTR(posSlash1% + 1, inData$, slash)

'  Se esiste uno slash, legge il mese ed il giorno della
'  data.

   IF posSlash1% <> 0 THEN
      mese% = VAL(inData$)
      giorno% = VAL(MID$(inData$, posSlash1% + 1))

'  Se esiste il secondo slash, legge l'anno della data.
'  Se l'anno è minore di 100, aggiunge 1900.

      IF posSlash2% <> 0 THEN
         anno% = VAL(MID$(inData$, posSlash2% + 1))
         IF anno% < 100 THEN anno% = anno% + 1900

'  L'anno deve essere compreso tra il 1900 e il 2500 inclusi.

         possAnno% = (anno% >= 1900 AND anno% <= 2500)

'  Se l'utente ha indicato solo il mese e il giorno, inserisce
'  l'anno corrente per completare la data.
      ELSE
         anno% = VAL(RIGHT$(DATE$, 4))
         possAnno% = TRUE
      END IF

'  Verifica la validità della data fornita.

      possGiorMe% = (giorno% > 0 AND giorno% <= GiornoMese%(mese%, anno%))
      possData% = possAnno% AND possGiorMe%
   ELSE
      possData% = FALSE
   END IF


'  Ritorna un valore pari a 0 nel parametro anno% se la
'  data non è accettabile.

   IF NOT possData% THEN anno% = 0
END SUB

SUB IndCorr (dataCorr&)

'  Il sottoprogramma IndCorr aggiunge una nuova voce
'  all'indice Agenda, mette in ordine l'indice e lo salva
'  sul disco.
'  Se l'indice contiene più di una voce, crea un
'  vettore temporaneo per memorizzare la versione
```

```
' precedente dell'indice. Copia la versione precedente
' in questo vettore.

  IF contImm% > 1 THEN
     REDIM Indice(contImm% - 1) AS immInd
     FOR imm% = 1 TO contImm% - 1
        Indice(imm%) = indData(imm%)
     NEXT imm%
  END IF

' Ridimensiona il vettore globale dell'indice per
' sistemare la nuova immissione.

  REDIM indData(contImm%) AS immInd

' Copia la prima versione dell'indice nel vettore
' dell'indice.

  IF contImm% > 1 THEN
     FOR imm% = 1 TO contImm% - 1
        indData(imm%) = Indice(imm%)
     NEXT imm%
  END IF

' Aggiunge il nuovo indice alla fine del vettore e
' riordina l'indice stesso.

  indData(contImm%).posData = contImm%
  indData(contImm%).dataLung = dataCorr&
  OrdIndData

' Salva la nuova versione dell'indice sul disco dove
' sarà disponibile per la successiva esecuzione del
' programma.

  OPEN "AGENDA.NDX" FOR OUTPUT AS #2

  FOR imm% = 1 TO contImm%
     WRITE #2, indData(imm%).dataLung, indData(imm%).posData
  NEXT imm%
  CLOSE #2

END SUB

SUB SalvGior (numRec%)

' Il sottoprogramma SalvGior salva su disco il record
'    nuovo o modificato nel database Agenda.
' Definisce una variabile record temporanea per eseguire
' l'operazione di salvataggio.
```

```
   DIM giorno AS recGior

'  Copia le informazioni dell'agenda dai due vettori
'  globali chiamati matt e pomer.

   giorno.matt7 = matt(7)
   giorno.matt8 = matt(8)
   giorno.matt9 = matt(9)
   giorno.matt10 = matt(10)
   giorno.matt11 = matt(11)
   giorno.matt12 = matt(12)
   giorno.pomer1 = pomer(1)
   giorno.pomer2 = pomer(2)
   giorno.pomer3 = pomer(3)
   giorno.pomer4 = pomer(4)
   giorno.pomer5 = pomer(5)
   giorno.pomer6 = pomer(6)

'  Scrive il record del giorno nel file e poi chiude
'  il database Agenda.

PUT #1, numRec%, giorno
CLOSE #1

END SUB

FUNCTION CercData% (dataFiss&)

'  La funzione CercData% cerca una data fissata nell'indice
'     Agenda; se la trova, ritorna la posizione del record
'     corrispondente nel database Agenda, se non la trova,
'     ritorna un valore pari a 0, il che significa che non
'     esiste un record dell'agenda per questa data.

   dataTrov% = 0

   pos1% = 1
   pos3% = contImm%

'  Eseguire una ricerca binaria, dividendo a metà
'  il vettore ordinato finché non si trova la data fissata
'  o non si vede che è assente dalla lista.

   DO WHILE dataTrov% = 0 AND pos1% <= pos3%
      pos2% = (pos1% + pos3%) \ 2

      SELECT CASE dataFiss&
         CASE indData(pos2%).dataLung
            dataTrov% = indData(pos2%).posData
```

```
'  Se la data non è stata trovata, decide quali
'  successive divisioni eseguire nella lista.

         CASE IS > indData(pos2%).dataLung
            pos1% = pos2% + 1
         CASE ELSE
            pos3% = pos2% - 1
      END SELECT

   LOOP

   CercData% = dataTrov%
END FUNCTION

SUB OrdIndData

'  OrdIndData riorganizza la lista dell'indice sulla base de
'     campo dataLung, un campo intero lungo che
'     rappresenta le date che al momento sono incluse nel
'     database Agenda.

'  Non cerca di riordinare se il conteggio del record è
'  1 o 0.

   lungh% = contImm%
   IF lungh% <= 1 THEN EXIT SUB

'  Usa l'algoritmo di ordinamento Shellsort.

   salt% = 1
   DO
      salt% = salt% * 2
   LOOP UNTIL salt% > lungh%

   DO WHILE salt% > 1
   salt% = (salt% - 1) \ 2
   DO
      ordin% = TRUE
      FOR prim% = 1 TO lungh% - salt%
         sec% = prim% + salt%
         data1& = indData(prim%).dataLung
         data2& = indData(sec%).dataLung
         IF data1& > data2& THEN
            SWAP indData(prim%), indData(sec%)
            ordin% = FALSE
         END IF
      NEXT prim%
    LOOP UNTIL ordin%
   LOOP

END SUB
```

Capitolo 31

Il programma GestioneDate

Questo esempio, chiamato programma *GestioneDate* descrive due diversi tipi di gestione degli eventi: una gestione del TIMER e una varietà di gestione di eventi generati da tastiera. Nella Parte VI di questo libro si può trovare una completa descrizione della gestione degli eventi. Il compito di *GestioneDate* è quello di rendere il programma *Agenda* più facile da usare. *Agenda* (il programma precedente presentato nel Capitolo 30) gestisce un calendario in cui si possono registrare gli appuntamenti di lavoro e personali per qualsiasi data presente o futura. *Agenda* è in grado di funzionare in modo autonomo ed è progettata per essere compilata come un file EXE e poi essere attivata direttamente dal prompt del DOS. Quando si avvia il programma, solitamente si inserisce una stringa che rappresenta la data del calendario che si desidera esaminare; ad esempio:

```
C>AGENDA 11/11/90
```

Come risposta, il programma *Agenda* fornisce un foglio dell'agenda a pieno schermo in cui si possono inserire gli appuntamenti per le varie ore del giorno. Il programma *GestioneDate* è un'interfaccia utente per *Agenda*, che offre un modo più vantaggioso di selezionare le date del calendario. Dopo la compilazione, il programma può essere attivato direttamente dal prompt del DOS, ma non richiede nessun argomento della linea di comando. Ogni volta che si avvia *GestioneDate*, il programma attiva una o più esecuzioni del programma *Agenda*. *Agenda* e *GestioneDate* sono progettati per rimanere su file separati del disco, ma insieme formano una sola applicazione.

ATTIVAZIONE DEL PROGRAMMA

Quando si attiva per la prima volta *GestioneDate*, il programma visualizza sullo schermo tre diverse informazioni. Nella parte superiore del video appare il mese corrente, ed appena sotto vi sono molte linee di istruzioni che riassumono i comandi da tastiera che si possono usare con questo programma. Infine, nella parte inferiore, vi è un orologio digitale aggiornato al secondo. Ecco un esempio di come si presenta lo schermo:

```
                 Dicembre 1990

          D    L    M    M    G    V    S
         --------------------------------
                                        1
          2    3    4    5    6    7    8
          9   10   11   12   13   14   15
         16   17   18   19   20   21   22
         23   24   25   26   27   28   29
         30   31

------------------------------------------------
    Seleziona una data e Visualizza l'agenda:
------------------------------------------------

←→                 Evidenziare un nuovo giorno.
↑↓                 Seleziona una nuova settimana.
Ctrl-← Ctrl-→      Seleziona un nuovo mese.
Ctrl-↑ Ctrl-↓      Seleziona un nuovo anno.

Invio              Visualizza l'agenda del giorno.
Ctrl-Invio         Uscire.

------------------------------------------------

                   4:59:27 p.m.
```

Nel calendario di un mese che appare nella parte superiore dello schermo, la data corrente viene evidenziata in reverse. Come suggerito nelle istruzioni, si possono usare i quattro tasti freccia della tastiera, talvolta insieme al tasto Ctrl, per selezionare la data con cui si desidera lavorare. Ecco una spiegazione estesa delle funzioni a tastiera di questo programma:

- Premere il tasto freccia verso destra (→) per spostare l'evidenziazione in avanti di un giorno oppure il tasto freccia verso sinistra (←) per spostarla indietro di un giorno.

- Premere il tasto freccia verso il basso (↓) per spostarsi in avanti di una settimana (ad esempio, dall'undicesimo al diciottesimo giorno del mese) oppure il tasto freccia verso l'alto (↑) per spostarsi indietro di una settimana (dall'undicesimo al quarto giorno).
- Premere Ctrl-→ per spostarsi sul successivo mese del calendario (ad esempio, da luglio ad agosto) oppure Ctrl -← per spostarsi sul precedente (da luglio a giugno). *GestioneDate* sostituisce il mese visualizzato con quello nuovo selezionato.
- Premere Ctrl-(↓) per spostarsi in avanti di un anno (ad esempio, da luglio 1989 a luglio 1990) oppure Ctrl-(↑) per spostarsi indietro di un anno (da luglio 1989 a luglio 1988).

L'evidenziazione in reverse rispetto allo sfondo identifica la data fissata. Una volta utilizzati i tasti freccia per evidenziare la data che si desidera selezionare, premere il tasto Invio. *GestioneDate* attiva il programma *Agenda*, inviandogli la data selezionata. *Agenda* visualizza immediatamente sullo schermo il calendario del giorno. Non ci sono cambiamenti rispetto al modo di lavorare proprio di *Agenda*. Si può visualizzare il calendario del giorno, immettere nuovi appuntamenti o modificare quelli già esistenti. Una volta terminato di lavorare con il giorno fissato, basta premere il tasto Q. Se sono stati fatti dei cambiamenti nell'agenda del giorno, il programma chiede una conferma per salvare i cambiamenti. Successivamente il programma *GestioneDate* riassume il controllo, visualizzando sullo schermo il calendario, le istruzioni e l'orario. Durante un'attivazione del programma, è possibile selezionare e lavorare con tutte le date che si desiderano. Ogni volta che si seleziona una data, *GestioneDate* esegue un'altra chiamata al programma *Agenda*. Tutti i nuovi appuntamenti immessi nelle varie date durante l'attivazione corrente verranno salvate su disco nel database AGENDA.DAT.

LA STRUTTURA DEL PROGRAMMA

GestioneDate crea anche una gestione degli eventi per rispondere alle istruzioni dell'utente da tastiera. Questo processo richiede tre principali fasi:

1. Una serie di istruzioni KEY che definiscono i tasti dedicati per la gestione degli eventi.
2. Una serie di istruzioni ON KEY che identificano le procedure di gestione degli eventi.
3. Una serie di istruzioni KEY(n) ON che attivano la gestione degli eventi da tastiera.

Inoltre, il programma usa le istruzioni ON TIMER e TIMER ON per attivare una gestione degli eventi che aggiorna l'orologio digitale:

```
ON TIMER(1) GOSUB VisualOra
TIMER ON
```

La procedura di gestione degli eventi per la gestione TIMER si trova nell'etichetta *VisualOra*. Una volta che queste gestioni degli eventi sono state definite ed attivate, un ciclo DO si assume l'operazione. Il compito principale di questo ciclo DO è di controllare i cambiamenti nei valori delle due variabili logiche centrali del programma:

- La variabile *chiam%* diventa vera se l'utente preme il tasto Invio per visualizzare la data selezionata al momento.
- La variabile *eseg%* diventa vera quando l'utente preme Ctrl-Invio per terminare il programma. In risposta ad un valore *chiam%* vero, il programma pone una data nella variabile *comLin$* e poi usa l'istruzione SHELL per attivare il programma *Agenda*:

```
SHELL "AGENDA" + comLin$
```

Quest'istruzione presuppone che il file AGENDA.EXE esista su disco nella directory corrente o nel percorso di ricerca definito con PATH. Dopo una chiamata al programma *Agenda*, *GestioneDate* ripristina sullo schermo il calendario, le istruzioni e l'orario digitale. Il ciclo continua, offrendo ripetute opportunità di lavorare con nuove date, finché l'utente preme Ctrl-Invio per terminare il programma. A questo punto, una routine di gestione degli eventi passa il valore *eseg%* a vero e il ciclo DO è finito:

```
LOOP UNTIL eseg%
```

Come tutti i principali programmi di questo libro, *GestioneDate* comprende molti commenti, così che si può studiare il programma da soli. In particolare, si possono esaminare le procedure di gestione degli eventi e notare le varie tecniche con cui ogni routine esegue il proprio compito. Ad esempio, si esamini come la routine *VisualOra* visualizzi il nuovo orario in un formato a.m./p.m. quando viene chiamata e come le routine *GiorSucc* e *GiorPrec* eseguano il loro compito di spostare la data selezionata in avanti o indietro.

IL LISTATO DEL PROGRAMMA GESTIONEDATE

```
' GESTIONEDATE.BAS
' Fornisce una vantaggiosa interfaccia utente per il
' programma Agenda. Aiuta l'utente a selezionare una
' data.

' Nella metà superiore dello schermo, GestioneDate
' visualizza il calendario di un mese, con il giorno
' corrente evidenziato in reverse. Nella metà
' inferiore, il programma presenta le istruzioni per
' selezionare una nuova data. L'utente può premere i
' quattro tasti freccia, da soli o insieme al tasto Ctrl,
' per selezionare un nuovo giorno, mese o anno. Una volta
' scelta la data, l'utente preme il tasto Invio per
' visualizzare e/o modificare gli appuntamenti del
' giorno. Il programma GestioneDate attiva il programma
' Agenda, inviando al programma la data selezionata
' dall'utente come argomento della linea di comando.

' AGENDA.EXE deve essere disponibile nella directory
' corrente perché l'esecuzione abbia successo.

' Dopo un'esecuzione del programma Agenda, GestioneDate
' riprende il controllo e consente all'utente di
' selezionare ogni ulteriore sequenza di date del
' calendario. Il programma è terminato quando l'utente
' preme il tasto Ctrl-Invio.

DECLARE SUB CancBuff ()
DECLARE SUB VisualAiu ()
DECLARE SUB VisualMese (mese%, giorno%, anno%)
DECLARE FUNCTION NumGior% (mese%, giorno%, anno%)
DECLARE FUNCTION NumData$ (mese%, giorno%, anno%)
DECLARE FUNCTION AnnoBisest% (anno%)
DECLARE FUNCTION Mese% (numMese%)
```

```
DECLARE FUNCTION GiorniMese% (mese%, anno%)

CONST MAXANNO% = 2100      ' L'ultimo anno possibile.
CONST MINANNO% = 1900      ' Il primo anno possibile.
CONST FALSE% = 0           ' Il valore logico falso.
CONST TRUE% = NOT FALSE%   ' Il valore logico vero.

'  Inizializza le tre variabili intere che rappresentano
'  la data corrente: meseCorr%, giorCorr% e annoCorr%.

meseCorr% = VAL(LEFT$(DATE$, 2))
giorCorr% = VAL(MID$(DATE$, 4, 2))
annoCorr% = VAL(RIGHT$(DATE$, 4))

'  Visualizza le istruzioni del programma nella metà
'  inferiore dello schermo.

CLS
VIEW PRINT 12 TO 22
VisualAiu

'  Visualizza il mese corrente nella metà superiore
'  dello schermo ed evidenzia la data corrente.

VisualAiu meseCorr%, giorCorr%, annoCorr%

'  Assegna i valori dei "tasti dedicati" ai numeri KEY
'  definiti dall'utente, da 15 a 24.

noMaius$ = CHR$(0)   ' Valore nullo.
ctrl$ = CHR$(4)      ' Tasto-Ctrl.

KEY 15, ctrl$ + CHR$(72)     ' Ctrl-SU
KEY 16, ctrl$ + CHR$(75)     ' Ctrl-SINISTRA
KEY 17, ctrl$ + CHR$(77)     ' Ctrl-DESTRA
KEY 18, ctrl$ + CHR$(80)     ' Ctrl-GIU
KEY 19, noMaius$ + CHR$(72)  ' SU
KEY 20, noMaius$ + CHR$(75)  ' SINISTRA
KEY 21, noMaius$ + CHR$(77)  ' DESTRA
KEY 22, noMaius$ + CHR$(80)  ' GIU
KEY 23, noMaius$ + CHR$(28)  ' INVIO
KEY 24, ctrl$ + CHR$(28)     ' Ctrl-INVIO

'  Definisce ed attiva la gestione degli eventi da tastiera.

ON KEY(15) GOSUB AnnoPrec    ' Ctrl-SU
KEY(15) ON

ON KEY(16) GOSUB MesePrec    ' Ctrl-SINISTRA
KEY(16) ON
```

```
ON KEY(17) GOSUB MeseSucc    ' Ctrl-DESTRA
KEY(17) ON

ON KEY(18) GOSUB AnnoSucc    ' Ctrl-GIU
KEY(18) ON

ON KEY(19) GOSUB SettPrec    ' SU
KEY(19) ON

ON KEY(20) GOSUB GiorPrec    ' SINISTRA
KEY(20) ON

ON KEY(21) GOSUB GiorSucc    ' DESTRA
KEY(21) ON

ON KEY(22) GOSUB SettSucc    ' GIU
KEY(22) ON

ON KEY(23) GOSUB ChiamCal    ' INVIO
KEY(23) ON

ON KEY(24) GOSUB EsciProg    ' Ctrl-INVIO
KEY(24) ON

'  Definisce ed attiva la gestione degli eventi TIMER,
'  per modificare l'orologio dopo ogni secondo.

ON TIMER(1) GOSUB VisualOra
TIMER ON

'  Inizializza i valori logici di controllo: eseg%
'  diventa vera quando l'utente preme Ctrl-Invio e chiam%
'  quando l'utente preme Invio.

eseg% = FALSE%
chiam% = FALSE%

'  Controllare i cambiamenti eseguiti nei valori di eseg%
'  o chiam%

DO
   IF chiam% THEN

'  Se chiam% diventa vera, attiva il programma Agenda.

      comLin$ = STR$(meseCorr%) + "/"
      comLin$ = comLin$ + STR$(giorCorr%) + "/"
      comLin$ = comLin$ + STR$(annoCorr%)
      CancBuff
      SHELL "AGENDA" + comLin$
```

```
'  Quando l'esecuzione di Agenda è completa, rivisualizza il
'  calendario del mese "corrente" e le istruzioni.

      VIEW PRINT
      CLS
      VIEW PRINT 12 TO 22
      VisualAiu

      VisualMese meseCorr%, giorCorr%, annoCorr%

'  Ritorna chiam% al valore falso per l'iterazione
'  successiva del ciclo.

      chiam% = FALSE%
   END IF

'  Il ciclo finisce quando eseg% diventa vera come risultato
'  del comando a tastiera inviato dall'utente: Ctrl-Invio.

LOOP UNTIL eseg%

VIEW PRINT
CLS
CancBuff
END

'  Il sottoprogramma VisualOra visualizza l'orario corrente
'  in fondo allo schermo.

VisualOra:
   VIEW PRINT
   LOCATE 23, 12

'  Converte l'orario su 24 ore nel formato a.m./p.m.

   oraCorr% = VAL(LEFT$(TIME$, 2))
   minSec$ = RIGHT$(TIME$, 6)
   SELECT CASE oraCorr%
      CASE 0
         PRINT "12"; minSec$; " a.m."
      CASE 1 TO 11
         PRINT USING "##& a.m."; oraCorr%; minSec$
      CASE 12
         PRINT TIME$ + " p.m."
      CASE ELSE
         PRINT USING "#& p.m."; oraCorr% - 12; minSec$
   END SELECT
RETURN

'  Il sottoprogramma GiorPrec seleziona la data del giorno
```

```
'  precedente rispetto a quello corrente, quando l'utente
'  preme il tasto freccia verso sinistra.

GiorPrec:
   IF giorCorr% > 1 THEN
      giorCorr% = giorCorr% - 1
   ELSEIF meseCorr% > 1 THEN
      meseCorr% = meseCorr% - 1
      giorCorr% = GiorniMese%(meseCorr%, annoCorr%)
   ELSE
      annoCorr% = annoCorr% - 1
      meseCorr% = 12
      giorCorr% = 31
   END IF
   VisualMese meseCorr%, giorCorr%, annoCorr%
RETURN

'  Il sottoprogramma GiorSucc seleziona la data del giorno
'  successivo rispetto a quello corrente, quando l'utente
'  preme il tasto freccia verso destra.

GiorSucc:
   IF giorCorr% < GiorniMese%(meseCorr%, annoCorr%) THEN
      giorCorr% = giorCorr% + 1
   ELSEIF meseCorr% <> 12 THEN
      giorCorr% = 1
      meseCorr% = meseCorr% + 1
   ELSE
      giorCorr% = 1
      meseCorr% = 1
      annoCorr% = annoCorr% + 1
   END IF
   VisualMese meseCorr%, giorCorr%, annoCorr%
RETURN

'  Il sottoprogramma SettPrec seleziona la data della
'  settimana precedente rispetto a quella corrente, quando
'  l'utente preme il tasto freccia verso l'alto.

SettPrec:
   IF giorCorr% > 7 THEN
      giorCorr% = giorCorr% - 7
      VisualMese meseCorr%, giorCorr%, annoCorr%
   END IF
RETURN

'  Il sottoprogramma SettSucc seleziona la data della
'  settimana successiva rispetto a quella corrente, quando
'  l'utente preme il tasto freccia verso il basso.
```

```
SettSucc:
   IF giorCorr% <= GiorniMese%(meseCorr%, annoCorr%) -7 THEN
      giorCorr% = giorCorr% + 7
      VisualMese meseCorr%, giorCorr%, annoCorr%
   END IF
RETURN

'  Il sottoprogramma MesePrec seleziona la data del mese
'  precedente rispetto a quello corrente, quando l'utente
'  preme il tasto Ctrl-Sinistra.

MesePrec:
   IF meseCorr% > 1 THEN
      meseCorr% = meseCorr% - 1
   ELSE IF annoCorr% > MINYEAR% THEN
      annoCorr% = annoCorr% - 1
      meseCorr% = 12
   END IF
   giorMax% = GiorniMese%(meseCorr%, annoCorr%)
   IF giorCorr% > giorMax% THEN giorCorr% = giorMax%
   VisualMese meseCorr%, giorCorr%, annoCorr%
RETURN

'  Il sottoprogramma MeseSucc seleziona la data del mese
'  successivo rispetto a quello corrente, quando l'utente
'  preme il tasto Ctrl-Destra.

MeseSucc:
   IF meseCorr% < 12 THEN
      meseCorr% = meseCorr% + 1
   ELSE IF annoCorr% < MAXYEAR% THEN
      annoCorr% = annoCorr% + 1
      meseCorr% = 1
   END IF
   giorMax% = GiorniMese%(meseCorr%, annoCorr%)
   IF giorCorr% > giorMax% THEN giorCorr% = giorMax%
   VisualMese meseCorr%, giorCorr%, annoCorr%
RETURN

'  Il sottoprogramma AnnoPrec seleziona la data dell'anno
'  precedente rispetto a quello corrente, quando l'utente
'  preme il tasto Ctrl-Su.

AnnoPrec:
   IF annoCorr% > MINYEAR% THEN
      annoCorr% = annoCorr% - 1
      giorMax% = giorniMese%(meseCorr%, annoCorr%)
      IF giorCorr% > giorMax% THEN giorCorr% = giorMax%
      VisualMese meseCorr%, giorCorr%, annoCorr%
   END IF
```

```
RETURN

'  Il sottoprogramma AnnoSucc seleziona la data dell'anno
'  successivo rispetto a quello corrente, quando l'utente
'  preme il tasto Ctrl-Giu.

AnnoSucc:
   IF annoCorr% < MAXYEAR% THEN
      annoCorr% = annoCorr% + 1
      giorMax% = GiorniMese%(meseCorr%, annoCorr%)
      IF giorCorr% > giorMax% THEN giorCorr% = giorMax%
      VisualMese meseCorr%, giorCorr%, annoCorr%
   END IF
RETURN

'  Il sottoprogramma ChiamCal passa il valore di chiam%
'  quando l'utente preme il tasto Invio.

ChiamCal:
   chiam% = TRUE%
RETURN

'  Il sottoprogramma EsciProg passa il valore di eseg%
'  quando l'utente preme Ctrl-Invio.

EsciProg:
   eseg% = TRUE%
RETURN

SUB CancBuff

'  La procedura CancBuff cancella tutti i caratteri dal
'     buffer della tastiera.

   DO WHILE INKEY$ <> ""
   LOOP

END SUB

FUNCTION NumData& (mese%, giorno%, anno%)

'  La funzione NumData& ritorna un intero lungo che
'     rappresenta la data specificata negli argomenti. Il
'     valore pari a 1 rappresenta: 1 Gennaio 1900.

   annoIniz% = 1900
   Genn% = 1
   GiorPerAn& = 365
```

```
'  Se la funzione riceve una data non valida,
'  il valore di ritorno è zero.

   tropPr% = (anno% < annoIniz%)
   meseErr% = (mese% < 1 OR mese% > 12)
   giorErr% = (gio% < 1 OR gio% > GiorniMese%(mese%, anno%))
   IF tropPr% OR meseErr% OR giorErr% THEN
      NumData& = 0
      EXIT FUNCTION
   END IF

'  Diversamente, calcolare i giorni trascorsi dal 1900.
   num& = giorPerAn& * (anno% - annoIniz%)

'  Aggiungere i giorni extra per gli anni bisestili.

   FOR annoCorr% = annoIniz% TO anno% - 1 STEP 4
      IF AnnoBisest%(annoCorr%) THEN num& = num& + 1
   NEXT annoCorr%

'  Conta i giorni dei mesi - fino
'  al mese specificato escluso.

   FOR MeseCorr% = Genn% TO mese% - 1
      num& = num& + GiorniMese%(meseCorr%, anno%)
   NEXT meseCorr%

'  Aggiunge i giorni fino alla data specificata e
'  ritorna il valore ottenuto.

   num& = num& + giorno%
   NumData& = num&

END FUNCTION

FUNCTION NumGior% (mese%, giorno%, anno%)

'  La funzione NumGior% ritorna un valore da 1 a 7, che rappresenta
'     il giorno della settimana (da domenica a sabato) della data
'     specificata negli argomenti. Una data non valida risulta in
'     un valore di ritorno pari a zero.

   g& = numData&(mese%, giorno%, anno%)
   IF g& <> 0 THEN gds% = g& MOD 7 + 1 ELSE gds% = 0

   NumGior% = gds%

END FUNCTION

FUNCTION AnnoBisest% (anno%)
```

```
' La funzione AnnoBisest% ritorna un valore booleano che
' indica se un determinato anno è bisestile.

   divPer4% = (anno% MOD 4 = 0)
   secolo% = (anno% MOD 100 = 0)
   secolo400% = (anno% MOD 400 = 0)

   AnnoBisest% = divPer4% AND (secolo% IMP secolo400%)

END FUNCTION

FUNCTION GiorniMese% (numMese%, anno%)

' La funzione GiorniMese% ritorna un intero da 28 a 31,
'    che indica il numero di giorni in un determinato
'    mese. L'argomento anno% viene richiesto per
'    determinare il numero dei giorni di febbraio.

' Ritorna un valore pari a zero per un mese non valido.

   IF numMese% < 1 OR numMese% > 12 THEN
      GiorniMese% = 0
      EXIT FUNCTION
   END IF

' Altrimenti, determina il numero dei giorni nel mese.

   SELECT CASE numMese%

' Feb.

      CASE 2
         giorni% = 28
         IF AnnoBisest%(anno%) THEN giorni% = giorni% + 1
' Apr, Giu, Set e Nov.

      CASE 4, 6, 9, 11
         giorni% = 30
' I mesi restanti.

      CASE ELSE
         giorni% = 31
   END SELECT

   GiorniMese% = giorni%

END FUNCTION

FUNCTION Mese$ (numMese%)
```

```
' La funzione Mese$ prende un argomento da 1 a 12 e
'    ritorna il nome del mese corrispondente. Dato un
'    numero non valido di mese, la funzione ritorna una
'    stringa vuota.

         m$ = "Gen   Feb   Mar   Apr "
   m$ =  m$ + "Mag   Giu   Lug   Ago "
   m$ =  m$ + "Set   Ott   Nov   Dic "

   IF numMese% >= 1 AND numMese% <= 12 THEN
      strMe$ = MID$(m$, (numMe% - 1) * 10 + 1, 10)
      strMe$ = RTRIM$(strMe$)
   ELSE
      strMe$ = ""
   END IF
   Mese$ = strMe$

END FUNCTION

SUB VisualAiu

' Il sottoprogramma VisualAiu visualizza sullo schermo le
'    opzioni a tastiera del programma.

' Codici ASCII per le quattro frecce:
   CONST sin% = 27
   CONST des% = 26
   CONST su% = 24
   CONST giù% = 25

' Visualizza il titolo ed una riga di lineette.

   tit$ = "Seleziona una data e visualizza l'agenda:"
   linTit$ = STRING$(LEN(tit$), "-")
   PRINT linTit$
   PRINT tit$
   PRINT linTit$

' Visualizza le istruzioni.

   PRINT CHR$(sin%); " "; CHR$(des%);
   PRINT "     Seleziona un nuovo giorno."
   PRINT CHR$(su%); " "; CHR$(giù%);
   PRINT "     Seleziona una nuova settimana."
   PRINT "Ctrl-"; CHR$(sin%); " Ctrl-"; CHR$(des%);
   PRINT "     Seleziona un nuovo mese."
   PRINT "Ctrl-"; CHR$(su%); " Ctrl-"; CHR$(giù%);
   PRINT "     Seleziona un nuovo anno."
   PRINT "Invio     ";
   PRINT "Visualizzare l'agenda del giorno."
```

```
   PRINT "Ctrl-Invio ";
   PRINT "Uscire."
   PRINT linTit$

END SUB

SUB VisualMese (mese%, giorno%, anno%)

'  La procedura VisualMese visualizza il mese del
'     calendario specificato negli argomenti.

'  Determina il mese, il giorno d'inizio ed il numero di giorni.

   primGior% = NumGior%(mese%, 1, anno%)
   lunghMese% = GiorniMese%(mese%, anno%)
   strMe$ = Mese$(mese%)

   TIMER OFF
   VIEW PRINT 1 TO 11
   CLS

'  Visualizza il nome del mese e le abbreviazioni del
'  giorno.
   PRINT SPACE$(14 - LEN(strMe$) \ 2); strMe$; anno%
   PRINT
   PRINT " D L M M G V S"
   PRINT " "; STRING$(26, "-")

'  Visualizza i giorni in colonna.
   dataCorr% = 0
   DO
      PRINT " ";
      FOR giorCorr% = 1 TO 7
         PrimGior% = (dataCorr% = PrimGior% AND dataCorr% = 0)
         NelMese% = (dataCorr% > 0 AND dataCorr% < lunghMese%)
         IF PrimGior% OR NelMese% THEN
         dataCorr% = dataCorr% + 1
         PRINT " ";
         IF dataCorr% = giorno% THEN COLOR 0, 7
         PRINT USING "##"; dataCorr%;
            COLOR 7, 0
         ELSE
            PRINT "  ";
         END IF
      NEXT giorCorr%
      PRINT
   LOOP UNTIL dataCorr% = lunghMese%

   TIMER ON

END SUB
```

Capitolo 32

Il programma DateStoriche

Il programma *DateStoriche* è un gioco a quiz che prova le conoscenze del giocatore riguardo le più importanti date storiche. Il programma visualizza le domande sullo schermo, una alla volta e concede al giocatore un numero fisso di secondi per rispondere. Ogni quesito è costituito dalla descrizione di un evento storico e di una lista di quattro possibili date. Il compito del giocatore è quello di scegliere la data corretta nel corso del tempo a sua disposizione. Dopo una risposta, il giocatore può continuare con un'altra domanda oppure uscire dal gioco quando vuole. Alla fine del gioco, il programma visualizza sullo schermo il punteggio raggiunto dal giocatore. Le domande fatte da *DateStoriche* sono memorizzate su disco in un file di testo chiamato DATESTORICHE.DAT. Bisogna preparare questo file prima di giocare. Un'opzione particolare del programma *DateStoriche* aiuta a creare il file delle domande; per eseguire il programma in questo modo, basta immettere il seguente comando dal prompt del DOS:

```
C>DATESTORICHE /Q
```

Il programma inizia un dialogo d'input che richiede l'inserimento della descrizione degli eventi (ognuna costituita al massimo da 65 caratteri) e la data corrispondente ad ogni evento. Ecco un esempio di come il programma salva queste informazioni nel file DATESTORICHE.DAT:

```
"Conquista Normanna dell'Inghilterra.                              "
1066
```

```
"Incoronazione di Napoleone Bonaparte.                          "
1804
"Scoperta dell'oro in California.                               "
1848
"Atterraggio degli Americani sulla luna.                        "
1969
```

Si può usare l'opzione /Q del programma *DateStoriche* per salvare nel file DATESTORICHE.DAT tutte le domande che si vogliono; inoltre in ogni momento si possono aggiungere altre domande al file. (In alternativa, è possibile anche usare un qualsiasi programma di elaborazione testi per scrivere il file.) *DateStoriche* è progettato per illustrare due strumenti di programmazione QuickBASIC: le istruzioni e le funzioni che producono la grafica sullo schermo e le istruzioni che danno effetti sonori speciali e musica. Inoltre, *DateStoriche* contiene alcuni interessanti esempi di funzioni matematiche per calcolare i valori trigonometrici ed i numeri casuali. Si possono trovare frammenti di questo programma in più parti del testo.

ATTIVAZIONE DEL PROGRAMMA

Una volta creato un file EXE dal codice sorgente del programma *DateStoriche* ed un file DATESTORICHE.DAT che contiene le domande, il quiz stesso può essere giocato in due modi. Nel modo di default, il programma, nel corso del gioco, produce vari effetti sonori. Nel modo "muto", gli effetti sonori vengono sostituiti da semplici immagini grafiche che rappresentano gli eventi del gioco. Per giocare nel modo di default basta immettere il nome del programma dal prompt del DOS:

```
C>DATESTORICHE
```

Il programma passa immediatamente nel modo grafico a colori (SCREEN 9, disponibile su schede grafiche EGA e VGA) e visualizza sullo schermo la prima domanda. Nella parte superiore del video appare la descrizione di un evento storico in una linea, e sotto ad essa, sulla sinistra, il programma presenta una lista con quattro possibili date. Sulla destra vi è, invece, un timer a dieci secondi che inizia a battere il tempo non appena visualizzata la domanda, e la seconda lancetta dell'orologio inizia a girare. Il giocatore deve scegliere una delle quattro date, premendo da tastiera i tasti A, B, C o D,

prima che i dieci secondi a disposizione siano trascorsi. Una volta selezionata la risposta, il programma avvisa se la risposta è corretta o errata. Inoltre, emette una o due brevi sequenze di note musicali che il giocatore imparerà a conoscere come reazione a risposte corrette o errate. Se la risposta è sbagliata, il programma visualizza quella giusta sullo schermo. Nel caso in cui l'utente non riesca a scegliere una risposta entro il tempo a disposizione, quando la seconda lancetta ha terminato il proprio giro suona un allarme. Il programma poi indica sullo schermo la risposta corretta. Nella parte inferiore del video, alla fine di ogni sequenza di domande appare questo messaggio:

```
Premere la barra spaziatrice per continuare
oppure il tasto Q per uscire dal gioco.
```

In base alla risposta del giocatore, il programma o visualizza un'altra domanda e azzera l'orologio oppure termina il gioco e visualizza il punteggio ottenuto. Per giocare in modo muto, immettere il seguente comando dal prompt del DOS:

```
C>DATESTORICHE /M
```

Come risultato, il programma non emette alcun effetto sonoro: l'orologio è silenzioso e non c'è nessuna musica dopo ogni risposta. Il programma, invece, visualizza una semplice immagine grafica in conseguenza delle risposte corrette o errate: una faccia o sorridente o triste. Le Figure 1 e 2 mostrano come appare lo schermo dopo ogni tipo di risposta.
Infine, si può modificare il tempo a disposizione: basta inserire l'opzione /T*n* dal prompt del DOS quando si attiva il programma. Il valore di *n* indica il numero di secondi consentiti; sono validi i numeri 5, 10, 15 o 20. Ad esempio, attivare il programma in questo modo per avere a disposizione 5 secondi per ogni risposta:

```
C>DATESTORICHE /T5
```

La Figura 3 indica come si presenta lo schermo con questa opzione. Si noti che il timer scandisce il tempo solo per cinque secondi.

Figura 1 Schermata dopo una risposta esatta

Figura 2 Schermata dopo una risposta errata

Figura 3 Schermata con l'orologio che scandisce i 5 secondi

LA STRUTTURA DEL PROGRAMMA

Il programma *DateStoriche* inizia leggendo la linea di comando dal prompt del DOS e determinando se sono state richieste speciali opzioni. La procedura *LinDos* esegue questo compito. Se l'utente inserisce l'opzione /Q nella linea di comando, il programma esegue una chiamata alla procedura *AggDom*, che instaura il dialogo d'input per aggiungere altre domande al file DATESTORICHE.DAT. Dopo questo dialogo, l'esecuzione del programma termina, ed il gioco non viene eseguito. Altrimenti, il programma passa al modo schermo 9. (Si noti che la versione corrente di questo programma può essere avviata solo su un sistema dotato di una scheda EGA, VGA o MCGA. Per funzionare con una scheda CGA, il programma richiede modifiche e revisioni.) Una chiamata alla procedura *LeggiDom* legge le domande del gioco in un vettore dei record chiamato *infData* ed una chiamata a *DomMis* sistema le domande in ordine casuale. Poi un lungo ciclo DO assume il controllo per porre le domande fino a quando il giocatore non vuole uscire o le domande sono finite. All'inizio di ogni sequenza, una chiamata alla procedura *AzzOrol* azzera l'orologio e lo visualizza sullo schermo. Il programma definisce ed attiva anche una gestione degli eventi ON TIMER

per aggiornare l'orologio dopo ogni secondo. La routine di gestione degli eventi, chiamata *ClicSeg;* si occupa di muovere la lancetta dei secondi e di emettere un ticchettio udibile (a meno che il programma sia attivato in modo "muto"). La procedura *VisualDom* visualizza sullo schermo la domanda e la lista di più risposte. Una chiamata alla procedura *PrepRisp* dà una lista casuale di risposte possibili ma errate insieme a quella corretta. In seguito, una chiamata alla funzione *Risp%* aspetta la scelta del giocatore e la visualizza sul video. Una volta ricevuta la risposta, la procedura *VisualDom* produce la giusta sequenza di note musicali o visualizza una faccia sorridente o triste, in base al modo corrente. Una chiamata alla procedura *Music* esegue una musica o una chiamata alla procedura *Faccia* disegna una faccia sullo schermo. Infine, il programma chiede al giocatore di premere la barra spaziatrice per continuare con un'altra domanda oppure di uscire premendo il tasto Q. Alla fine del gioco, una chiamata alla procedura *Risult* mostra sul video il punteggio del giocatore. Sparsi nel libro vi sono altri esempi di istruzioni grafiche QuickBASIC. Si può imparare di più sulle operazioni grafiche eseguite dal programma concentrandosi su tre routine: le procedure *Faccia* e *AzzOrol* ed il sottoprogramma di gestione degli eventi *ClicSeg*. Insieme, queste procedure usano molte istruzioni descritte nella Parte IV di questo libro, comprese SCREEN, VIEW, WINDOW, COLOR, PAINT, CIRCLE, DRAW, LINE e PSET.

IL LISTATO DEL PROGRAMMA DATESTORICHE

```
' DATESTORICHE.BAS
' Conduce un gioco a quiz che mette alla prova le
' conoscenze del giocatore a proposito di date storiche.

' Questo programma visualizza sullo schermo le domande,
' una alla volta. Ognuna è costituita dalla descrizione
' di un evento storico in una linea e da quattro date
' (indicate con le lettere A, B, C e D), di cui solo una
' è corretta. Per scegliere una risposta, il giocatore
' preme il tasto di una delle quattro lettere e il programma
' dice al giocatore se la risposta è giusta.

' Per default, il giocatore ha a disposizione 10 secondi
' per rispondere ad ogni domanda; un orologio nell'angolo in
' basso a destra dello schermo indica quanto tempo rimane.
```

```
' Dopo ogni risposta, il giocatore può scegliere se continuare
' (premendo la barra spaziatrice) o smettere di giocare (premendo
' il tasto Q). Quando il gioco è terminato, il programma visualizza
' il numero delle risposte corrette ed il punteggio in percentuale.

' Il programma DateStoriche legge le domande da un file di testo
' chiamato DATESTORICHE.DAT, memorizzato su disco nella stessa
' directory come il programma stesso. È possibile creare questo
' file prima che si inizi il gioco, DateStoriche può essere d'aiuto
' in ciò quando si avvia il programma dal prompt del DOS:

'                       C>DATESTORICHE /Q

' Invece di giocare, l'opzione /Q instaura un dialogo
' d'input che pone le domande per il gioco. Il programma
' poi le salva in un formato appropriato nel file
' DATESTORICHE.DAT. Si possono sempre aggiungere altri
' quesiti a questo file, riattivando il programma con
' l'opzione /Q. Ogni volta che si gioca, il programma
' legge il file — fino ad un massimo di 200 domande — e
' sistema le domande in ordine casuale.

' Un'altra opzione che si può dare dal prompt del DOS
' determina come il programma reagisce alle risposte corrette
' ed errate. Per default, il programma esegue due diversi temi
' musicali per distinguere le risposte giuste da quelle sbagliate.
' Per attivare il programma nel modo "muto", occorre usare la
' seguente opzione dal prompt del DOS:

'             C>DATESTORICHE /M

' Ciò inibisce tutti gli effetti sonori del programma. In
' questo modo, il programma visualizza una faccia triste
' o una sorridente in conseguenza di una risposta
' sbagliata o giusta.

' Infine, una terza opzione dal prompt del DOS consente al
' giocatore di specificare il numero di secondi per rispondere ad
' ogni domanda. L'opzione è rappresentata da /Tn, dove n è il
' numero di secondi — un valore pari a 5, 10, 15 o 20.

DECLARE SUB AggDom ()
DECLARE SUB VisualDom (numDom%, rispEsat%, altra%)
DECLARE SUB Faccia (att%)
DECLARE SUB AzzOrol ()
DECLARE SUB DomMis ()
DECLARE SUB Music (esat%)
DECLARE SUB PrepRisp (listDate%(), dataEsat%, posEsat%)
DECLARE SUB LeggiLinDos (nuoDom%)
DECLARE SUB LeggiDom ()
```

```
DECLARE SUB Risult (dom%, rispEsat%)
DECLARE FUNCTION Risp% ()
DECLARE FUNCTION Rad (inAng%)
DECLARE FUNCTION InterCas% (min%, max%)

CONST domMax% = 200    ' Numero massimo di domande in
                       ' DATESTORICHE.DAT.
CONST lunghLanc% = 6   ' Lunghezza della lancetta dei
                       ' secondi dell'orologio.
CONST angIniz% = 90    ' Punto d'inizio dell'orologio.
CONST angFin% = -270   ' Punto finale dell'orologio.


CONST sorr% = 0        ' Codice da inviare alla procedura Faccia per
                       ' rappresentare un sorriso.
CONST tris% = 2        ' Codice da inviare alla procedura Faccia per
                       ' rappresentare un'espressione triste.
CONST FALSE = 0        ' Valore logico falso.
CONST TRUE = NOT FALSE ' Valore logico vero.

'  La struttura recData definisce il tipo di record per il
'  vettore infData — la lista di domande.

TYPE recData
   even AS STRING * 65
   data AS INTEGER
END TYPE

'  Il vettore infData memorizza la lista di domande.

DIM SHARED infData(domMax%) AS recData

'  La variabile numRec% rappresenta il numero di domande
'  che il programma ha letto da DATESTORICHE.DAT. La
'  variabile logica tempFin% diventa vera quando è scaduto
'  il tempo per rispondere ad una data domanda. La variabile logica
'  muto% è vera nel modo muto del programma.

COMMON SHARED numRec%, tempFin%, muto%

'  La variabile sec% indica il numero di secondi concessi
'  per rispondere. La variabile incrAng% è l'angolo di incremento
'  corrispondente ad un tic dell'orologio.

COMMON SHARED sec%, incrAng%

RANDOMIZE TIMER      ' Inizializza il generatore di numeri casuali.

LeggiLinDos%         ' Legge le opzioni della linea di comando del DOS.
IF aggSol% THEN END  ' Non si gioca se l'opzione è /Q.
```

```
SCREEN 9
COLOR 1, 15
LeggiDom    ' Legge il file DATESTORICHE.DATE.
DomMis      ' Dispone le domande in ordine casuale.

'  Gioca.

totEsat% = 0
i% = 0
DO
' Visualizza l'orologio e fà partire il ticchettio.
' Inoltre, inizializza le variabili che regolano il
' ticchettio.

   AzzOrol
   ang% = angIniz%
   xCirc = 0
   yCirc = lunghLanc%

'  Visualizza una domanda e riceve la risposta
'  dell'utente. Conta il numero di domande e di risposte
'  esatte date.

   i% = i% + 1
   VisualDom i%, esat%, contin%
   IF esat% THEN totEsat% = totEsat% +1

   tempFin% = FALSE
   TIMER STOP
LOOP UNTIL i% >= numRec% OR NOT contin%

TIMER OFF

'  Visualizza su schermo i risultati del gioco.

SCREEN 0
Risult i%, totEsat%

END

'  La routine ClicSeg muove la lancetta dei secondi in
'  avanti dopo ogni secondo. La gestione degli eventi che
'  chiama questa routine si trova nel sottoprogramma
'  AzzOrol.

ClicSeg:
   VIEW (399, 174)-(639, 349)
   WINDOW (-10, -10-(10, 10)

'  Cancella la lancetta dei secondi dalla sua precedente posizione.
```

```
LINE (0, 0)-(xCirc, yCirc), 15

'  Calcola la nuova posizione e ridisegna la lancetta dei
'  secondi.

   ang% = ang% + incrAng%
   xCirc = lunghLanc% * COS(rad(ang%))
   yCirc = lunghLanc% * SIN(rad(ang%))
   LINE (0, 0)-(xCirc, yCirc), 4

'  Determina se è giunto il momento di rispondere.
   IF ang% > finAng% THEN
      IF NOT muto% THEN SOUND 50, .5
      CIRCLE (0, 0), .25, 1
      PAINT (0, 0), 1, 1
   ELSE
      IF NOT muto% THEN SOUND 1000, 10
      TIMER OFF
      tempFin% = TRUE
   END IF
RETURN

'  Il sottoprogramma NoFileDom viene chiamato se il
'  programma non trova il file DATESTORICHE.DAT su disco.

NoFileDom:

   SCREEN 0
   PRINT "Non riesco a trovare DATESTORICHE sul disco."
   PRINT "Devi creare questo file prima di iniziare a"
   PRINT "giocare. Come aiuto nella creazione del file,"
   PRINT "attivare il programma dal prompt del DOS come "
   PRINT "segue:"
   PRINT
   PRINT "    DATESTORICHE /Q"
   PRINT
   END

' Queste linee DATA contengono istruzioni DRAW per la
' faccia triste e per quella sorridente che il programma
' visualizza sullo schermo dopo una risposta corretta o
' errata.

DATA 4
DATA F1 D1 F1 D1 F1 D1 F1 D1 F1 D1 F5
DATA F1 R1 F1 R1 F1 R1 F1 R1 F1 R1 R10
DATA E1 R1 E1 R1 E1 R1 E1 R1 E1 R1 E5
DATA E1 U1 E1 U1 E1 U1 E1 U1 E1 U1

SUB AggDom
```

```
' Il sottoprogramma AggDom instaura un dialogo d'input
'    per aggiungere nuove domande al file
'    DATESTORICHE.DAT. (L'esecuzione del programma
'    termina dopo una chiamata a questa routine. Non si
'    gioca.) Ogni domanda inserita viene aggiunta alla
'    fine del file.

' Il record nuoDom rappresenta una domanda e la
' sua corretta risposta.
  DIM nuoDom AD recData

  CLS
  tit$ = "Aggiunge Nuove Domande al file DATESTORICHE.DAT"
  PRINT tit$
  PRINT STRING$(LEN(tit$), "-")
  PRINT

  PRINT "Ogni domanda è costituita dalla descrizione di un"
  PRINT "evento (può essere al massimo di 65 caratteri), e"
  PRINT "la data in cui è avvenuto l'evento."
  PRINT
  OPEN "DATESTORICHE:DAT" FOR APPEND AS #1

  DO
  PRINT "Immetti una descrizione dell'evento:"

' Visualizza una riga di trattini per facilitare la
' specifica della descrizione da parte dell'utente.

     FOR i% = 1 TO 65
        IF i% MOD 10 = 0 THEN PRINT "!"; ELSE PRINT "-";
     NEXT i%
     PRINT

     LINE INPUT nuoDom.even
     INPUT "Qual è la data di questo evento"; nuoDom.data
' Conferma che l'utente vuole salvare questa domanda.

     LOCATE , , 1
     PRINT "Salvo questa domanda? (S o N) ";
     DO
        sìNo$ = UCASE$(INKEY$)
     LOOP UNTIL sìNo$ <> "" AND INSTR("SN", sìNo$) <> 0
     PRINT sìNo$

     IF sìNo$ = "S" THEN
        WRITE #1, nuoDom.even
        WRITE #1, nuoDom.data
     END IF
```

```
'  Chiede se ci sono altre domande da immettere.

      PRINT
      PRINT "Altre domande? (S o N) ";
      DO
         sìNo$ = UCASE$(INKEY$)
      LOOP UNTIL sìNo$ <> "" AND INSTR("SN", sìNo$) <> 0
      PRINT sìNo$
      PRINT

   LOOP UNTIL sìNo$ = "N"
   CLOSE #1
END SUB

SUB VisualDom (numDom%, rispEsat%, altra%)

'  Il sottoprogramma VisualDom visualizza sullo schermo
'     una domanda ed aspetta per un numero determinato di
'     secondi la risposta del giocatore. Poi, questa
'     routine visualizza un messaggio che indica se la
'     risposta è corretta.

'  Il vettore listRisp% ha la lista delle quattro possibili
'  risposte.

   DIM listRisp%(4)

   VIEW
   WINDOW
'  Visualizza la domanda in una finestra nella parte
'  superiore dello schermo.
   LINE (47, 63)-(582, 92), 4, B
   PAINT (48, 64), 15, 4
   LOCATE 6, 8
   PRINT infData(numDom%).even

'  Visualizza le possibili risposte in una finestra sotto
'  la domanda.

   LINE (71, 159)-(170, 235), 4, B
   PAINT (72, 160), 15, 4

   scelt$ = "ABCD"

'  Prepara la lista delle risposte a scelta multipla.

   PrepRisp listRisp%(), infData(numDom%).data, rispEsat%

   FOR i% = 1 TO 4
      LOCATE 12 + i%, 12
```

```
      PRINT MID$(scelt$, i%, 1); "."; listRisp%(i%)
   NEXT i%

'  Chiama la funzione Risp% per aspettare la risposta del
'  giocatore e poi visualizza il messaggio appropriato.

   IF Risp% = rispEsat% THEN
      LOCATE 20, 10
      PRINT "Esatto!"
      IF muto% THEN Faccia (sorr%) ELSE Music (TRUE)
      rispEsat% = TRUE
'  La variabile tempFin% diventa vera quando il tempo a
'  disposizione del giocatore è finito.
   ELSEIF tempFin% THEN
      LOCATE 20, 10
      PRINT "Tempo scaduto! La risposta è";
      PRINT infData(numDom%).data
      rispEsat% = FALSE
   ELSE
      LOCATE 20, 10
      PRINT "Errato! La risposta è";
      PRINT infData(numDom%).data

'  Nel modo muto disegna una faccia oppure emette un suono.

      IF muto% THEN Faccia (trist%) ELSE Music (FALSE)
      rispEsat% = FALSE
   END IF

'  Dà al giocatore le istruzioni per continuare o
'  terminare il gioco.

   TIMER OFF
   LOCATE 22, 10
   PRINT "Premere la Barra Spaziatrice per continuare"
   LOCATE 23, 10
   PRINT "oppure il tasto 'Q' per uscire dal gioco."
   DO
      risp$ = UCASE$(INKEY$)
   LOOP UNTIL risp$ = " " OR risp$ = "Q"
   IF risp$ = "Q" THEN altra% = FALSE ELSE altra% = TRUE

END SUB

SUB Faccia (att%)

'  Il sottoprogramma Faccia disegna sul video una faccia
'     triste o una sorridente, in conseguenza ad una risposta
'     corretta o errata alla domanda corrente.
```

```
TIMER OFF

'  Disegna usando le istruzioni DRAW delle linee DATA.

   RESTORE
   disLin$ = ""
   READ infLin%
      FOR i% = 1 TO infLin%
      READ d$
      disLin$ = disLin$ + d$ + " "
   NEXT i%

   VIEW
   WINDOW

'  Disegna la faccia.

   PSET (339, 204)

'  Sistema la posizione e l'angolo del disegno se è
'  richiesta la faccia triste.

   IF att% = trist% THEN DRAW "BM+50,+15"
   DRAW "A=" + VARPTR$(att%) + " C4"
   DRAW disLin$

'  Disegna i cerchi per le circonferenze della faccia e
'  degli occhi.

   CIRCLE (355, 190), 3, 4
   CIRCLE (375, 190), 3, 4
   CIRCLE (365, 205), 40, 4

'  Riempie l'area della faccia.

   PAINT (365, 190), 15, 4

END SUB

FUNCTION Risp%

'  La funzione Risp% legge la risposta del giocatore alla
'     domanda corrente. La funzione ritorna un intero che
'     rappresenta la posizione della risposta del
'     giocatore  nella lista a più possibilità.

   VIEW
   WINDOW

'  Prepara una piccola finestra per visualizzare la
'  risposta del giocatore.
```

```
   LINE (210, 205)-(246, 227), 4, B
   PAINT (211, 206), 15, 4
   LOCATE 14, 27
   PRINT "Risposta:"
   LOCATE 16, 28
   PRINT " ";
   LOCATE , 29

'  Aspetta la risposta.

   inRisp% = 0
   DO

'  Smette di aspettare se il tempo è scaduto.

      IF tempFin% THEN EXIT DO
      inRisp$ = UCASE$(INKEY$)
      IF inRisp$ <> "" THEN inRisp% = INSTR("ABCD", inRisp$)
   LOOP UNTIL inRisp% <> 0

   PRINT inRisp$;
   Risp% = inRisp%

END FUNCTION

SUB AzzOrol

'  Il sottoprogramma AzzOrol visualizza sullo schermo
'     l'orologio ed attiva la gestione degli eventi per
'     chiamare il sottoprogramma ClicSeg.

'  Visualizza il nome del programma nella parte superiore
'  dello schermo.
   CLS 0
   tit$ = "Seleziona la data corretta"
   LOCATE 3, 40 - LEN(tit$) / 2
   PRINT tit$

'  Definisce la finestra grafica e stabilisce un sistema
'  opportuno di coordinate per disegnare l'orologio.

   VIEW (399, 174)-(639, 349)
   WINDOW (-10, -10)-(10, 10)

'Disegna l'orologio ed il pulsante al centro

   CIRCLE (0, 0), lunghlanc% - 1, 4
   PAINT (0, 0), 15, 4
   CIRCLE (0, 0), .25, 1
   PAINT (0, 0), 1, 1
```

```
'  Disegna la lancetta dei secondi nella posizione d'inizio.

   LINE (0, 0)-(0, lunghlanc%), 4
   altx% = (399 + PMAP(0, 0)) / 8
   alty% = (174 + PMAP(lunghlanc%, 1)) / 14
   LOCATE alty%, altx%
   PRINT USING "##"; sec%

'  Definisce ed attiva la gestione degli eventi.

   ON TIMER(1) GOSUB ClicSeg
   TIMER ON

END SUB

SUB DomMis

'  Il sottoprogramma DomMis risistema la lista delle
'     domande in ordine casuale. (Il programma in seguito
'     pone le domande in quest'ordine.)

   FOR i% = 1 TO numRec%

'  Scambia ogni domanda con un'altra in una posizione
'  casuale.

      j% = InterCas%(1, numRec%)
      SWAP infData(i%), infData(j%)
   NEXT i%

END SUB

SUB Music (esat%)

'  Il sottoprogramma Music suona due brevi temi musicali
'     differenti a seconda che la risposta dell'utente
'     sia corretta o meno.

   IF esat% THEN
      PLAY "o4 l8 e l16 f# e > l3 c"
   ELSE
      PLAY "o2 l8 e l16 f# e < l3 c"
   END IF

END SUB

SUB PrepRisp (lisData%(), dataEsat%, posEsat%)

'  Il sottoprogramma PrepRisp prepara il vettore di
'     quattro elementi delle risposte per ogni domanda.
```

```
'     Il programma "ritorna" i dati per indirizzo:
'     lisData% è la lista delle risposte; dataEsat% è la
'     risposta corretta e posEsat% è la sua posizione nel
'     vettore lisData%.

   CONST deltaData% = 25        ' L'intervallo delle risposte.

   rispMax% = UBOUND(lisData%)

'  Colloca temporaneamente la risposta esatta in prima posizione.

   posEsat% = 1
   lisData%(posEsat%) = dataEsat%

'  Determina le possibilità più antiche e più recenti di
'  risposte errate a seconda del valore di deltaData%.
'  Nessuna data errata deve essere posteriore a quella
'  corrente.

   dataCorr% = VAL(RIGHT$(DATE$, 4))
   IF dataEsat% + deltaData% > dataCorr% THEN
      dataMax% = dataCorr%
   ELSE
      dataMax% = dataEsat% + deltaData%
   END IF
      dataMin% = dataEsat% - deltaData%

'  Genera le date errate casuali, assicurandosi che non
'  ci siano date doppie nel vettore lisData%.

   FOR i% = 2 TO rispMax%
      DO
         centr% = FALSE
         lisData%(i%) = InterCas%(dataMin%, dataMax%)
         FOR j% = 1 TO i% - 1
            IF lisData%(i%) = lisData%(j%) THEN centr% = TRUE
         NEXT j%
      LOOP UNTIL NOT centr%
   NEXT i%

'  Ordina il vettore lisData% in ordine cronologico.

   FOR i% = 1 TO rispMax%
      FOR j% = i% + 1 TO rispMax%
         IF lisData%(i%) > lisData%(j%) THEN
            SWAP lisData%(i%), lisData%(j%)

'  Tiene memoria della posizione della risposta esatta.

            IF i% = posEsat% THEN
```

```
               posEsat% = j%
            ELSEIF j% = posEsat% THEN
               posEsat% = i%
            END IF
         END IF
      NEXT j%
   NEXT i%

END SUB

FUNCTION Rad (inAng%)

'  La funzione Rad converte in radianti l'ampiezza
'  di un angolo espressa in gradi.

   CONST PI = 3,14159
   Rad = (inAng% / 360!) * 2 * PI

END FUNCTION

FUNCTION InterCas% (min%, max%)

'  La funzione InterCas% ritorna un intero casuale tra
'     min% e max%, compresi.

   InterCas% = min% + INT(RND * (max% - min% + 1))

END FUNCTION

SUB LeggiLinDos (nuoDom%)

'  Il sottoprogramma LeggiLinDos legge le opzioni fornite
'     dall'utente sulla linea di comando del DOS.

'  L'opzione /M esegue il programma in modalità muta,
'  cioè senza effetti sonori.

   IF INSTR(COMMAND$, "/M") <> 0 THEN muto% = TRUE

'  L'opzione /Q esegue la procedura AggDom in modo che
'  l'utente possa aggiungere domande all'archivio del
'  quiz. Ciò non sottintende che si debba giocare.
   IF INSTR(COMMAND$, "/Q") <> 0 THEN nuoDom% = TRUE
   IF nuoDom% THEN AggDom

'  L'opzione /T consente all'utente di stabilire il tempo
'  massimo per la risposta. Il valore standard è 10 e le
'  altre possibili scelte sono 5, 15, 20.

   posTemp% = INSTR(COMMAND$, "/T")
```

```
   IF posTemp% <> 0 THEN
      sec% = VAL(MID$(COMMAND$, posTemp% + 2))
      SELECT CASE sec%
         CASE IS <= 0
            sec% = 10
         CASE 1 TO 7
            sec% = 5
         CASE 8 TO 12
            sec% = 10
         CASE 13 TO 17
            sec% = 15
         CASE IS > 17
            sec% = 20
      END SELECT
   ELSE
      sec% = 10
   END IF

'  Determina l'angolo di ogni scatto delle lancette sulla
'  base del tempo massimo per la risposta.

inAng% = (angFin% - angIniz%) / sec%

END SUB

SUB LeggiDom

'  La procedura LeggiDom legge le domande disponibili dal
'     file DATESTORICHE.DAT. Le domande sono memorizzate
'     nel vettore globale dei record chiamati InfData.

   ON ERROR GOTO NoFileDom
      OPEN "DATESTORICHE.DAT" FOR INPUT AS #1
   ON ERROR GOTO 0

   numRec% = 0
   DO WHILE NOT EOF(1) AND numRec% < domMax%
      numRec% = numRec% + 1
      INPUT #1, infData(numRec%).even
      INPUT #1, infData(numRec%).data
   LOOP
   CLOSE #1

END SUB

SUB Risult (dom%, rispEsat%)

'  Il sottoprogramma Risult visualizza i messaggi sullo
'     schermo alla fine di un dato gioco, riassumendo il
'     punteggio del giocatore.
```

```
   LOCATE 3, 10
   PRINT "Il tuo risultato:"
   LOCATE 4, 10
   PRINT STRING$(18, "-")
   LOCATE 5, 10
   PRINT USING "Hai risposto correttamente a ## domande"; rispEsat%;
   PRINT USING "su ##"; dom%;
   IF dom% <> 1 THEN PRINT "s" ELSE PRINT " "
   percent% = domEsat% * 100 / dom%
   LOCATE 6, 10
   PRINT USING "il punteggio è ###%."; percent%

END SUB
```

Appendice A

Il codice ASCII

0	null	27	←	54	6	81	Q
1	☺	28	cursor right	55	7	82	R
2	☻	29	cursor left	56	8	83	S
3	♥	30	cursor up	57	9	84	T
4	♦	31	cursor down	58	:	85	U
5	♣	32	spc	59	;	86	V
6	♠	33	!	60	<	87	W
7	bell ●	34	"	61	=	88	X
8	◘	35	#	62	>	89	Y
9	tab ○	36	$	63	?	90	Z
10	lf ■	37	%	64	@	91	[
11	home ♂	38	&	65	A	92	\
12	ff ♀	39	'	66	B	93	]
13	cr	40	(	67	C	94	^
14	♫	41	)	68	D	95	_
15	☼	42	*	69	E	96	`
16	►	43	+	70	F	97	a
17	◄	44	,	71	G	98	b
18	↕	45	-	72	H	99	c
19	!!	46	.	73	I	100	d
20	¶	47	/	74	J	101	e
21	§	48	0	75	K	102	f
22	_	49	1	76	L	103	g
23	↨	50	2	77	M	104	h
24	↑	51	3	78	N	105	i
25	↓	52	4	79	O	106	j
26	→	53	5	80	P	107	k

108	l	145	æ	182	╢	219	█
109	m	146	Æ	183	╖	220	▄
110	n	147	ô	184	╕	221	▌
111	o	148	ö	185	╣	222	▐
112	p	149	ò	186	║	223	▀
113	q	150	û	187	╗	224	α
114	r	151	ù	188	╝	225	β
115	s	152	ÿ	189	╜	226	Γ
116	t	153	Ö	190	╛	227	π
117	u	154	Ü	191	┐	228	Σ
118	v	155	¢	192	└	229	σ
119	w	156	£	193	┴	230	µ
120	x	157	¥	194	┬	231	τ
121	y	158	₧	195	├	232	Φ
122	z	159	ƒ	196	─	233	θ
123	{	160	á	197	┼	234	Ω
124	¦	161	í	198	╞	235	δ
125	}	162	ó	199	╟	236	∞
126	~	163	ú	200	╚	237	ø
127	⌂	164	ñ	201	╔	238	ε
128	Ç	165	Ñ	202	╩	239	∩
129	ü	166	ª	203	╦	240	≡
130	é	167	º	204	╠	241	±
131	â	168	¿	205	═	242	≥
132	ä	169	⌐	206	╬	243	≤
133	à	170	¬	207	╧	244	⌠
134	á	171	½	208	╨	245	⌡
135	ç	172	¼	209	╤	246	÷
136	ê	173	¡	210	╥	247	≈
137	ë	174	«	211	╙	248	°
138	è	175	»	212	╘	249	•
139	ï	176	░	213	╒	250	·
140	î	177	▒	214	╓	251	√
141	ì	178	▓	215	╫	252	^{n}
142	Ä	179	│	216	╪	253	^{2}
143	Å	180	┤	217	┘	254	■
144	É	181	╡	218	┌	255	

Indice Analitico

≡ Allievo del C.I.P.

FIORENZA ANNA MARIA

VIA CRONATO, 94
95123 CATANIA

INFORMATICA

I programmatori in Basic, professionisti e non troveranno in questo utile manuale una trattazione completa di tutti i 200 comandi e funzioni del QuickBasic 4.5, rivisti e ordinati per una comoda consultazione. La particolarità di questo volume sta nell'essere organizzato per argomenti-operazioni, illustrando la struttura dei programmi, i grafici, le stringhe, etc. In questo modo è possibile focalizzarsi sulla sintassi necessaria oppure approfondire le proprie conoscenze di programmazione. Ogni argomento importante viene illustrato con un programma di esempio. In particolare, vengono trattati tra i vari argomenti i tipi di dati, le variabili e le strutture di dati, le funzioni e le operazioni di sistema.

L. 59.000

tecniche nuove